MASTER EN SCIENCES INFORMATIQUES

Approche transformationnelle du développement des systèmes experts

Une démarche d'analyse des besoins en ALBERT et de spécification abstraite de solution en KIF d'un problème de planification

Blondiau, Jean-François

Award date:
1993

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix à Namur
Institut d'Informatique

APPROCHE TRANSFORMATIONNELLE DU DEVELOPPEMENT DES SYSTEMES EXPERTS

Une démarche d'analyse des besoins en ALBERT et de spécification abstraite de solution en KIF d'un problème de planification.

Jean-François Blondiau

Mémoire rédigé en vue de l'obtention du diplôme de Licencié et Maître en Informatique

Promoteur : Eric Dubois

Année académique 1992-1993

RESUME

Dans le cadre de recherches menées pour concevoir un langage et une démarche de spécification abstraite des systèmes experts calquée sur le génie logiciel classique, ce document propose d'une part une analyse des besoins en ALBERT (langage d'analyse des besoins orienté-objet et basé sur le concept d'agent), d'un système expert chargé de résoudre le célèbre problème du singe et des bananes, et d'autre part une démarche de spécification abstraite du même problème en utilisant KIF (langage de représentation abstraite des connaissances d'un système expert).

Outre ces deux spécifications, nous proposerons une esquisse de transformation d'ALBERT à KIF, ainsi qu'une évaluation de KIF en tant que langage de spécification abstraite, et une solution personnelle de spécification (en KIF) de la métaconnaissance du problème.

ABSTRACT

Within the framework of researches conducted in the design of a language and a method supporting the specification of expert systems, adopting a software engineering point of view, this thesis aims at making distinct the requirement analysis phasis (using the ALBERT language based on the concept of agent) and the design phasis (using the KIF language based on abstract knowledge representation). The whole approach is illustrated through the handling of the well-known "monkey & bananas" problem.

On top of the two specifications, we sketch some transformations making possible a smooth transition from ALBERT to KIF. Finally, we evaluate KIF as a design language and we propose a possible way of incorporating metaknowledge in KIF.

REMERCIEMENTS

Je tiens en premier lieu à remercier Michel Lemoine, responsable de mon stage à Toulouse, avec qui j'ai noué des liens qui vont au delà du rapport stagiaire/maître de stage. Il est à l'origine du sujet de ce mémoire et il en a suivi le développement de bout en bout. Ce fut un plaisir de partager son bureau pendant les quelques mois qu'a duré mon stage.

Je remercie aussi Eric Dubois, mon promoteur. Il m'a d'abord permis d'effectuer ce stage au CERT à Toulouse, puis à mon retour a pris en charge la supervision du mémoire. Ses conseils avisés et les longs moments qu'il m'a consacrés malgré un emploi du temps très chargé en ont fait un promoteur précieux, que je recommande aux étudiants qui doivent encore choisir le leur.

Aussi bien à Toulouse qu'à Namur, plusieurs personnes m'ont été d'un grand secours dans l'élaboration de ce document. A Toulouse, je tiens à remercier Jack Foisseau et Jean-Lou Bussenot pour leur soutien de grande valeur, ainsi que toute l'équipe du DERI et du GIA pour leur accueil. A Namur, je remercie Philippe Du Bois et Michaël Petit pour leurs explications claires, ainsi que Frédéric Dubru pour le temps qu'il a passé avec moi sur ALBERT, de manière tout à fait désintéressée.

Enfin, je remercie mon père pour son soutien moral et financier durant les quelques mois qu'a duré la rédaction de ce document.

TABLE DES MATIERES

INTRODUCTION

1. LE PROBLEME

L'un des grands domaines de la recherche en Intelligence Artificielle concerne le développement des **systèmes experts** ou **systèmes de connaissances**. Il s'agit de systèmes informatiques qui simulent le raisonnement humain dans des domaines d'expertise précis, comme le diagnostic médical ou la planification de mouvement en robotique.

Actuellement, le développement des systèmes experts se passe encore de manière fort empirique. Les ingénieurs qui conçoivent ces systèmes pensent très vite en termes d'implémentation suivant le type de langage dans lequel le système sera décrit (langages logiques, comme Prolog; fonctionnels comme LISP; algorithmiques comme C; etc.)

Comme dans les systèmes d'informations classiques, ce développement trop vite orienté vers l'implémentation pose deux problèmes :

- un problème de *portabilité* : s'il faut changer de langage d'implémentation du système, il faut recommencer la conception du système depuis le début.
- un problème d'*évolution* : l'ajout de nouvelles fonctionnalités ou la correction d'erreurs s'avèrent épineux si le système n'a pas été proprement développé.

Ces problèmes ressemblent à s'y méprendre à ceux que l'on rencontre dans le développement de gros logiciels classiques, lorsqu'on néglige l'application d'un cycle de développement correct à ceux-ci. L'élaboration de ces cycles de développement est du ressort du *génie logiciel* ("software engineering").

Il serait intéressant d'appliquer les méthodes du génie logiciel au développement des systèmes experts. Actuellement, la tendance des recherches en génie logiciel est de développer les applications suivant une approche **transformationnelle** : à partir des cahier des charges (informel), élaborer une spécification abstraite du problème (aussi appelée *analyse des besoins*), puis élaborer successivement des spécifications de plus en plus raffinées par transformation, c'est-à-dire en utilisant des techniques de transformation correctes à une spécification pour produire la suivante, de façon à pouvoir prouver de manière formelle la correspondance de ces spécifications. La dernière étape du processus de transformation est l'implémentation du logiciel (voir figure 0.1.).

Qui dit spécification formelle dit langage formel de spécification. En génie logiciel, des langages de spécification comme Z (basé sur la théorie mathématique des ensembles) ou VDM (basé sur la logique des prédicats) ont fait leurs preuves.

Par contre, un tel langage n'a pas encore été défini pour la spécification formelle des systèmes de connaissances.

Le projet KSL (Knowledge Specification Language), développé au Centre d'Etudes et de Recherches de Toulouse depuis l'année 1992, vise à développer ce langage formel de spécification abstraite des systèmes de connaissances.

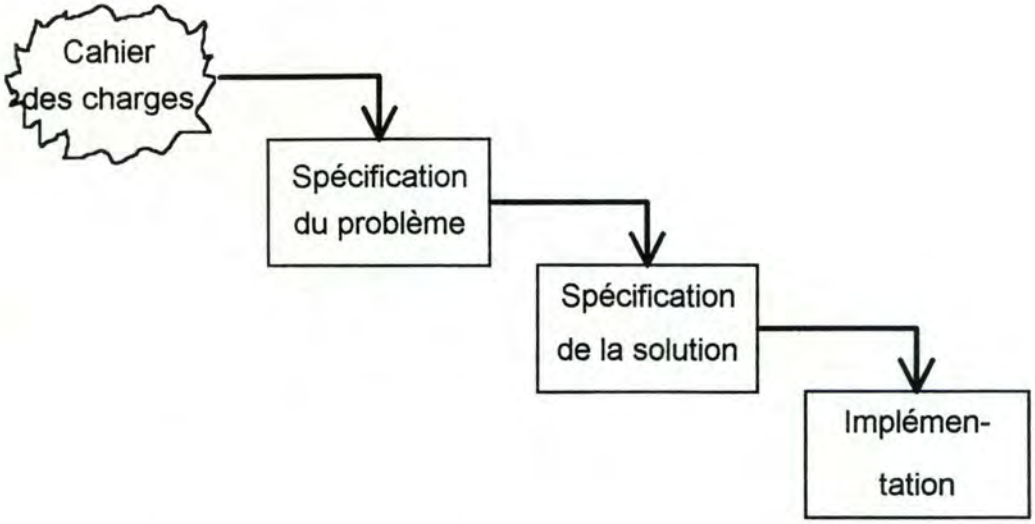

Figure 0.1. Cycle de développement transformationnel d'un logiciel.

La première phase de ce projet consiste en une étude de l'existant, c'est-à-dire notamment des langages de spécification classiques dont nous venons de parler.

La première orientation de ce mémoire est d'étudier un langage nommé KIF qui, s'il n'est pas conçu en soi comme un langage de spécification des systèmes de connaissances , permet au moins une représentation abstraite des connaissances de ces systèmes. Nous verrons dans quelle mesure il est possible d'utiliser les concepts de KIF pour élaborer une spécification abstraite de solution pour un système de connaissances.

Dans le processus de spécification transformationnel, on distingue deux grandes étapes : l'analyse des besoins, qui correspond à poser formellement le problème) et la spécification de la solution (ou *conception*) qui correspond à spécifier formellement la solution du problème. La frontière entre les deux est parfois floue : ainsi, un langage de spécification comme VDM propose une démarche de transformation complète (depuis le niveau le plus abstrait d'analyse des besoins jusqu'à une spécification très proche de l'implémentation choisie). D'autre part, certains langages sont spécifiquement orientés "analyse des besoins", d'autres sont orientés "conception", d'autres enfin sont partagés entre les deux. Comme langage de spécification, KIF appartient plutôt à cette catégorie "mixte". Il serait donc intéressant de poser le problème correctement, sous forme d'une analyse des besoins, avant de le spécifier en KIF[1].

La deuxième orientation de ce mémoire consiste donc à présenter une spécification du problème du singe et des bananes à ce niveau élevé d'abstraction, le niveau de l'*ingénierie des besoins* ("requirement engineering"). Ce niveau correspond à la description du système composite, c'est-à-dire de ses agents et de leurs interactions, indépendamment de toute considération d'ordre logiciel. Nous utiliserons le langage ALBERT, qui est conçu dans ce sens, pour élaborer cette analyse des besoins.

---

[1]Notons une fois pour toutes que le terme "spécification" appliqué à KIF signifiera toujours "spécification de la solution".

Le but de ce document, à travers ces deux orientations, est de proposer une étude par la pratique de ces deux étapes du cycle de vie d'un système expert, et de contribuer ainsi à l'élaboration d'un "génie des connaissances", analogue au génie logiciel classique.

2. CONTENU

REMARQUE : Afin de permettre une double lecture de ce document, nous avons conçu les exposés relatifs à ALBERT et à KIF de manière indépendante. Ce choix est dû au fait que certains lecteurs peuvent être plus intéressés par l'aspect "analyse des besoins" et donc par l'analyse des besoins du problème du singe en ALBERT, tandis que d'autres peuvent préférer l'étude de KIF et son évaluation, indépendamment d'une représentation plus abstraite du problème du singe. Le lien entre l'analyse des besoins en ALBERT et la spécification formelle en KIF est établi à la fin du chapitre 4.

Le premier chapitre présente brièvement la notion de système de connaissances, et décrira les concepts relatifs aux systèmes de planification, qui sont un cas particulier de système expert.

Le deuxième chapitre présente le problème du singe et des bananes de manière informelle (énoncé, contraintes, description des actions, première idée de représentation), puis proposera une démarche transformationnelle d'analyse des besoins en ALBERT.

Le troisième chapitre est consacré au langage KIF. Il en présente les objectifs et les concepts, afin de familiariser le lecteur avec le langage.

Le quatrième chapitre propose une spécification en KIF du système chargé de résoudre le problème du singe et des bananes. En fin de chapitre, nous proposons également une esquisse des transformations nécessaires pour passer du texte ALBERT au texte KIF.

Le cinquième chapitre présente une évaluation de KIF comme langage de spécification abstraite d'un système de connaissances, sur base de la spécification opérée au chapitre 4.

Le sixième chapitre, enfin, propose une solution de représentation de l'aspect "métaconnaissance" du problème du singe et des bananes (aspect occulté dans les deux chapitres qui précèdent) et évalue cette solution.

CHAPITRE 1
INTRODUCTION AUX SYSTEMES EXPERTS

Ce court chapitre a pour objectif de présenter très succinctement la notion de système expert, pour permettre au lecteur de se familiariser avec les concepts de ce type de système informatique particulier.

Nous présenterons de façon très globale les systèmes experts en général en décrivant les concepts relatifs à ces systèmes, en particulier de la représentation des connaissances dans un système expert, en évoquant les problèmes d'acquisition de ces connaissances, et en proposant une classification des systèmes experts et des problèmes qu'ils résolvent. Nous terminerons par une brève évocation des concepts relatifs à la planification, puisque le système que nous allons tenter de spécifier de façon abstraite dans la suite de ce document résout un problème typique de planification.

1.1. QUE SONT LES SYSTEMES EXPERTS?

Cette section et les deux suivantes sont inspirées de [JACKSON90].

1.1.1. Définition

Un **système expert** est un système informatique qui représente les connaissances d'un sujet spécialisé et raisonne sur base de ces connaissances.

Le système expert :

- simule le raisonnement humain dans un domaine d'expertise précis ;
- raisonne sur des représentations de la connaissance humaine ;
- distingue la base de connaissances (représentation) et le moteur d'inférence (raisonnement) ;

1.1.2. Concepts

Acquisition des connaissances

L'*acquisition des connaissances* est le transfert et la transformation de l'expertise de résolution potentielle d'un problème, depuis une source de connaissances vers un programme.

Cette acquisition passe par différentes étapes :

- **Identification** : identifier la classe de problèmes, les critères de solution, les ressources, les contraintes temporelles et financières ;
- **Conceptualisation** : découvrir les concepts fondamentaux et leurs interrelations (caractérisation des données, flux d'informations, structure sous-jacente du domaine) ;

- **Formalisation** : comprendre la nature de l'espace sous-jacent de recherches, vérifier le degré de certitude et la complétude de l'information;
- **Implémentation** : exprimer les règles sous forme exécutable, prendre des décisions relatives à la structure des données et au degré d'indépendance des modules du programme ;
- **Test** : évaluer le système expert sur de larges échantillons de test.

Cette acquisition passe par des interviews entre un expert et un ingénieur. Les problèmes rencontrés sont les suivants :

- l'expert utilise un jargon imprécis pour les non-initiés ;
- les faits et les principes rencontrés dans son domaine d'expertise ne se basent pas toujours sur un modèle déterministe ou sur une théorie mathématique ;
- il est extrêmement difficile de représenter la fiabilité des informations et les connaissances basées sur l'expérience plutôt que sur une formation formelle, en raison justement de l'aspect informel de cette expérience (p.ex., comment expliquer qu'un aviateur réussisse mieux ses atterrissages après plusieurs années d'expérience qu'au début de sa carrière, et comment formaliser cette expérience?) ;
- le contexte de la connaissance est plus large que le domaine d'expertise en soi (connaissance du monde de tous les jours, bon sens, ...).

Pour résoudre ces problèmes, des programmes d'acquisition automatique ont été mis au point.

Représentation des connaissances

Nous reviendrons sur le concept de représentation (section 1.2.).

Application des connaissances

Un système expert dispose d'une représentation de la **métaconnaissance** du problème, c'est-à-dire la connaissance sur la façon d'utiliser les connaissances. Cette métaconnaissance permet l'application de stratégies pour le contrôle et l'utilisation des connaissances.

Explication des solutions

Un système expert doit être capable d'expliquer comment il arrive à une solution (transparence vis-à-vis de l'utilisateur). Il doit donc être capable d'expliquer son exécution.

1.2. REPRESENTATION DES CONNAISSANCES

1.2.1. Définition et critères

Une **représentation** est un ensemble de conventions syntaxiques et sémantiques qui permettent de décrire des choses.

Une représentation a donc besoin d'un **langage**. Dans le cadre des systèmes experts, les critères d'un langage de représentation sont les suivants :

- **Adéquation logique** : possibilité de représenter toutes les distinctions de cas possibles (ce qui induit la présence de quantificateurs) ;
- **Puissance heuristique** : possibilité de résoudre les problèmes à partir de la représentation ;
- **Facilité dénotationnelle** : facilité d'écriture et de lecture, indépendamment de l'interprétation du langage.

1.2.2. Représentation des connaissances et systèmes experts

Connaissance procédurale et connaissance déclarative

C'est la principale distinction de représentation des connaissances d'un système expert. La *connaissance procédurale* d'un problème est de savoir *comment* résoudre ce problème ; la *connaissance déclarative* d'un problème est de savoir *ce qu'est* ce problème, ce qui le caractérise. Une description procédurale (pas à pas, algorithmique) correspond mieux à certains types de problèmes dans lesquels la méthode de résolution est connue, tandis qu'une description déclarative (description du but final et des contraintes de résolution) convient aux problèmes dont les contraintes de résolution sont trop dynamiques pour qu'il puisse être décrit procéduralement. La plupart des langages de représentation des connaissances d'un système expert sont déclaratifs (qu'ils soient logiques, comme PROLOG ou fonctionnels comme LISP).

Heuristiques et complexité

Un programme déterministe classique qui devrait résoudre le même problème qu'un système expert partirait de la situation de départ et générerait tous les états possibles jusqu'aux solutions finales. Certains problèmes sont d'une telle complexité qu'ils n'ont pas de solution algorithmique ou que cette solution n'est pas calculable sur machine. Les systèmes experts utilisent des **heuristiques**, c'est-à-dire des principes de résolution qui ne reposent sur aucune théorie fondée mais dont on sait qu'ils fonctionnent dans la plupart des cas, pour aboutir plus directement a une solution. Ces heuristiques ne garantissent évidemment pas que l'on obtienne toutes les solutions possibles, ni même que l'on génère la meilleure solution. Elles représentent la connaissance informelle de l'expert et constituent la métaconnaissance du problème, c'est-à-dire la connaissance des stratégies à appliquer à la connaissance brute du problème.

1.3. TYPOLOGIE

1.3.1. Classification des systèmes experts

Les systèmes experts peuvent être regroupés en classes suivant le type de problème qu'ils sont amenés à résoudre. Une classification généralement admise est celle que propose Peter Jackson ([JACKSON 90]) :

- les ***systèmes d'interprétation*** infèrent des descriptions de situation à partir d'observations (élucidation de structures chimiques, compréhension de signaux) ;
- les ***systèmes de prévision*** infèrent les conséquences probables de situations ou d'événements (prévisions météorologiques ou financières) ;
- les ***systèmes d'aide au diagnostic*** infèrent les fautes d'un système à partir des symptômes relevés (tâches dans le domaine médical, mécanique et électronique) ;
- les ***systèmes de design*** développent des configurations d'objets suivant certaines contraintes (par exemple, l'arrangement optimal de machines dans un espace confiné) ;
- les ***systèmes de planification*** génèrent des séquences d'actions en vue d'atteindre des buts donnés (planification des mouvements d'un robot, routage). Nous reviendrons plus en détail sur les concepts relatifs aux problèmes de planification ;
- les ***systèmes de monitoring*** étudient le comportement de systèmes en vue d'éviter des situations menaçantes (contrôle du trafic aérien, centrales nucléaires) ;
- les ***systèmes de déboguage*** génèrent des remèdes aux fautes d'un système (par exemple, aide a la programmation) ;
- les ***systèmes de réparation*** génèrent et administrent des remèdes aux fautes d'un système (remédiation à des défauts de fonctionnement d'un réseau d'ordinateurs, par exemple) ;
- les ***systèmes d'instruction*** (ou "systèmes auteurs") diagnostiquent et traitent les problèmes d'apprentissage d'étudiants dans un domaine particulier ;
- les ***systèmes de contrôle*** gouvernent le comportement d'un système en anticipant les problèmes, en planifiant des solutions et en contrôlant les actions nécessaires (gestion de bataille, contrôle de mission).

1.3.2. Problèmes de classification et problèmes de construction

On peut diviser les problèmes en deux grandes catégories : problèmes de classification et problèmes de construction.

Dans les ***problèmes de classification***, l'ensemble des solutions est fini et parfaitement déterminé. Le rôle du système expert est de déterminer parmi ces solutions, celle qui correspond aux données du problème. C'est typiquement le cas des systèmes de diagnostic. Le mécanisme d'inférence d'un tel système est appelé *correspondance heuristique* et se déroule en trois étapes :

- **abstraction des données** : généralisation des données brutes à des catégories de données (exemple: température du corps à 38° => température du corps supérieure à la normale) ;
- **correspondance heuristique** proprement dite : processus de mise en correspondance entre les données abstraites et une solution abstraite plutôt générale (température du corps supérieure a la normale => possibilité d'infection). Ce processus est dit *heuristique* parce que le lien entre les données abstraites et les solutions abstraites n'est pas biunivoque et que le choix d'une solution abstraite se fait suivant des règles heuristiques ;

- **raffinement de la solution** : recherche de la solution dans l'espace des solutions défini par la solution abstraite, en raisonnant sur les données brutes ou en demandant d'autres données.

Les ***problèmes de construction*** sont ceux dans lesquels l'espace des solutions est soit infini, soit trop large pour pouvoir être entièrement circonscrit. La solution est construite pas à pas, en tenant compte de contraintes sur le produit fini. C'est typiquement le cas des systèmes de planification ou de design.

Il faut noter que la distinction entre problèmes de classification et problèmes de construction n'est pas forcément nette. Ainsi, tous les systèmes de diagnostic ne fonctionnent pas obligatoirement par classification heuristique, parce que l'espace des solutions doit parfois pouvoir être enrichi par de nouvelles solutions obtenues par construction.

1.4. QUELQUES CONCEPTS RELATIFS AUX PROBLEMES DE PLANIFICATION

Un **état** du monde[2] est une vision instantanée de celui-ci à un moment donné. Un **comportement** est une suite d'états consécutifs du monde.

Le monde ne peut changer d'état que suite à l'occurrence d'un **événement**. Un événement est un fait exprimable comme par exemple "la chute des feuilles". L'état du monde dans lequel les feuilles sont sur les arbres est changé par cet événement, puisque l'état qui résulte de la chute des feuilles est celui dans lequel les arbres sont dénudés.

Une **action** est un type d'événement particulier, en ce qu'elle est *exécutée* par un **agent**, c'est-à-dire un intervenant actif (qui a le pouvoir de changer l'état du monde), et d'une manière intentionnelle. Dans le cas simple où il y a un seul agent et où ces actions sont déterministes, on peut les décrire en termes de fonctions d'état à état, ou **fonctions de changement d'état**.

Un agent exécute des actions afin de satisfaire un **but**. Un but peut être vu, soit comme un fait exprimable par une proposition logique ("Jean est à Bruxelles"), soit comme un ensemble d'états du monde dans lesquels cette proposition logique est vérifiée (c'est-à-dire tous les états dans lesquels Jean est à Bruxelles)[3].

Un **plan** est un ensemble d'actions[4] structuré, visant à atteindre un but à partir d'un état initial donné. L'**exécution** de ce plan produit une suite d'états (donc, un comportement) dont l'état final est un état dans lequel le but est atteint. Lorsqu'on n'est pas intéressé par l'historique d'un état, on peut simplement considérer que l'exécution d'un plan produit un état, qui est cet état final.

---

[2]La notion de monde est assez large : il peut s'agir en toute généralité du monde entier, comme il peut s'agir (dans un problème pratique) du domaine du problème, c'est à dire des objets qui le composent.

[3]Dans l'étude pratique qui va suivre, nous préférerons assimiler la notion de but à une proposition logique. Les états dans lesquels cette proposition sera vérifiée seront appelés **états-buts**.

[4]Plusieurs agents peuvent exécuter un même plan de façon coopérante. On parle alors de *planification multi-agents*.

Un **problème de planification** se pose donc en ces termes : connaissant un état initial (généralement, l'état présent du monde) et un but à atteindre, déterminer un plan dont l'exécution produit un état dans lequel ce but est atteint.

CHAPITRE 2
LE PROBLEME DU SINGE
DESCRIPTION ET ANALYSE DES BESOINS

2.1. INTRODUCTION

Le problème du singe et des bananes est un problème classique de planification, repris dans un grand nombre d'ouvrages traitant d'Intelligence Artificielle, en particulier de systèmes experts. Comme "cas d'école", il présente un bon compromis entre généralité et simplicité : il est assez complet pour donner une vision correcte des problèmes de planification en particulier et des systèmes experts en général, tout en restant d'un niveau suffisamment abordable pour servir d'exemple intuitif à tout exposé mettant en oeuvre la notion de système expert.

La prochaine section énoncera le problème et décrira les contraintes du problème ainsi que les actions possibles dans le système, en se basant sur [BUSSENOT92].

La section suivante est du ressort de ce qu'on appelle l'*ingénierie des besoins* (requirement engineering), c'est à dire l'ensemble des techniques permettant l'analyse des besoins du problème. Cette étape se veut une spécification abstraite du problème, dans laquelle sont présentés les agents intervenant dans la description du problème ainsi que les actions permettant de changer les états de ces agents. Pour opérer cette analyse des besoins, nous utiliserons le langage ALBERT, présenté dans [DUBOIS93]. Nous renvoyons le lecteur non familiarisé avec ALBERT à ce document.

2.2. DESCRIPTION DU PROBLEME

2.2.1. Enoncé

Dans une pièce carrée de dimension 10*10 se trouvent :

- d'une part, un singe ;
- d'autre part, plusieurs objets inanimés. Ces objets sont :
    - un lit
    - une échelle
    - un régime de bananes
    - une couverture

L'exemple classique de disposition du singe et des objets est le suivant :

- le lit est sur le sol
- l'échelle est posée sur le lit
- le régime de bananes est accroché au plafond

- la couverture est entre les mains du singe
- le singe lui-même est au sol

Le problème consiste à déterminer la séquence d'actions que le singe doit effectuer pour avoir en main le régime de bananes (et donc pouvoir le manger), et ce quelle que soit la configuration de départ (c'est à dire l'état global de la pièce et de ce qui se trouve dedans). Bien entendu, le singe n'est capable d'effectuer que certaines actions dans un ensemble restreint de possibilités. Ces actions sont :

- prendre un objet
- lâcher un objet
- aller en un point (x,y) de la pièce
- déposer un objet en un point (x,y) de la pièce
- monter sur un objet
- descendre d'un objet

De plus, il ne peut effectuer ces actions que sous certaines conditions. Celles-ci sont par exemple :

- le singe ne peut pas porter d'objets lourds
- il ne peut pas porter plus d'un objet à la fois
- il ne peut prendre un objet au plafond que s'il est monté sur échelle et si celle-ci se trouve en dessous de l'objet, donc à la même localisation horizontale

L'ensemble de ces contraintes, repris de [BUSSENOT92], est énuméré dans le point 2.2.2.

Une solution de ce problème dans le cas habituel décrit plus haut serait la séquence d'actions suivante :

1. le singe lâche la couverture ;
2. il va vers le lit ;
3. il monte sur le lit ;
4. il prend l'échelle ;
5. il descend du lit ;
6. il déplace l'échelle pour l'installer sous le régime de bananes ;
7. il monte sur l'échelle ;
8. il attrape le régime de bananes.

Afin d'être plus général, nous considérerons que le but du singe n'est pas nécessairement d'attraper les bananes, mais n'importe lequel des objets inanimés. Cet objet (l'objet cible) lui sera indiqué comme paramètre de l'énoncé.

2.2.2. Contraintes informelles du problème

Nous reprenons ici l'énumération des contraintes du problème telle qu'elle est définie dans [BUSSENOT92], à l'exception de celles relatives à la stratégie de résolution (celle-ci sera traitée au chapitre 6).

1. Il y a une seule échelle.
2. Le singe peut se déplacer seul.
3. L'échelle est assez grande pour que le singe puisse toucher le plafond une fois dessus, et assez légère pour être portée par le singe.
4. La taille des autres objets sera ignorée.

5. Le poids des objets physiques doit être représenté.
6. Le singe voit tous les objets dans la pièce.
7. Le singe peut accomplir certaines actions sans qu'il faille lui dire en détail comment faire.
8. Le système produira une solution en termes des actions précédentes.
9. La localisation d'un objet est un attribut de l'objet[5].
10. La localisation horizontale est un point 2D ; la valeur de la localisation verticale est sur-le-sol/au plafond/sur-objet(X) (ou "porté", voir contrainte 14).
11. Tous les objets physiques ont la même représentation ; l'objet-cible n'est pas distingué.
12. L'énoncé initial du problème précisera l'objet-cible.
13. Le système modifiera la localisation horizontale d'un objet à chaque fois que celui-ci sera déplacé.
14. La valeur "porté" pour la localisation verticale d'un objet signifie que cet objet est entre les mains du singe.
15. Pour saisir ou monter sur un objet, le singe doit être à la même localisation horizontale que celui-ci.
16. Le singe doit être sur l'échelle pour attraper un objet au plafond.
17. Quand un objet est lâché par le singe, sa position horizontale est celle du singe et sa position verticale est "sol".
18. Le singe peut descendre d'un objet tout en tenant un objet.
19. Le singe ne peut pas monter sur un objet s'il en tient un autre.
20. Le singe ne peut porter plus d'un objet à la fois.
21. Le système ne vérifiera pas la légalité des configurations.
22. Le système ne fournira pas d'aide à l'établissement de la configuration initiale.

2.2.3. Description informelle des actions

Nous reprenons ici la description proposée dans [BUSSENOT92]. Cette description est de la forme préconditions / postconditions.

Saisir-objet (objet)

PRECONDITIONS

- **objet** est un objet physique
- le singe ne porte rien
- **objet** a un poids léger
- **objet** et le singe ont la même position horizontale
- il n'y a rien sur **objet**
- le singe est sur le sol et **objet** n'est pas au plafond ou bien le singe est sur l'échelle et **objet** est au plafond

[5]Le terme "objet" utilisé dans cette contrainte désigne aussi le singe.

POSTCONDITIONS

- le singe porte **objet**
- la position verticale de **objet** est "porté"

Lâcher-objet (objet)

PRECONDITIONS

- **objet** est un objet physique
- le singe porte **objet**

POSTCONDITIONS

- le singe ne porte plus rien
- la position verticale de **objet** est le sol

Aller-en (xy)

PRECONDITION

- le singe est sur le sol

POSTCONDITION

- SI le singe ne porte rien ALORS
  - le singe est en **xy**
- SI le singe porte un objet ALORS
  - le singe est en **xy**
  - l'objet est en **xy**

Mettre-objet-en (objet, xy)

PRECONDITIONS

- préconditions de **aller-en (xy)**
- **objet** est un objet physique
- le singe tient **objet**

POSTCONDITIONS

- la position horizontale du singe est **xy**
- la position horizontale de **objet** est **xy**
- le singe ne porte plus rien
- la position verticale de **objet** est le sol

Monter-sur-objet (objet)

PRECONDITIONS

- **objet** est un objet physique
- le singe est sur le sol
- **objet** est sur le sol
- la position horizontale de **objet** est la même que celle du singe
- le singe ne porte rien

POSTCONDITION

- le singe est sur **objet**

Sauter-sur-sol

PRECONDITION
- le singe n'est pas sur le sol

POSTCONDITION
- le singe est sur le sol

2.2.4. Première idée de représentation

Afin de fixer les idées quant à la représentation du singe et des objets physiques, nous allons, à partir de l'énoncé du problème et des contraintes citées ci-dessus, proposer une représentation (informelle) possible du concept de singe et de celui d'objet physique sous forme d'une liste d'attributs.

Les objets physiques

L'énoncé du problème nous dit qu'on distingue quatre objets physiques dans la pièce : un régime de bananes, un lit, une couverture, une échelle. Le nom d'un objet identifie cet objet et doit donc être une composante de sa représentation.

D'autre part, la contrainte 9 nous dit que la localisation d'un objet est un attribut de l'objet. Cette localisation peut être éclatée en deux parties :

- une position horizontale par rapport au sol : celle-ci est, d'après la contrainte 10, un point 2D que l'on peut donc représenter par un couple (abscisse, ordonnée) où abscisse et ordonnée sont des valeurs entières comprises entre 1 et 10 (taille de la pièce).
- une position verticale : d'après la contrainte 10, elle peut être "sol" (l'objet est au sol), "plafond" (l'objet est accroché au plafond) ou "sur-objet (X)" (l'objet est posé sur l'objet désigné par X). Cette position verticale peut donc être représentée par une des valeurs suivantes : *{sol, plafond, porté, nom d'objet}*, où *nom d'objet* est l'un des noms qui identifient un objet physique.

Enfin, la contrainte 5 nous dit que le poids des objets physiques doit être représenté. Ce poids permet de regrouper les objets en deux catégories : ceux qui sont assez légers pour être portés par le singe, et ceux qui sont trop lourds pour lui. Dès lors, on peut représenter ce poids par une des valeurs *{lourd, léger}*.

En résumé, nous pouvons associer à un objet physique donné les attributs suivants :

- un **nom** *{bananes, lit, couverture, échelle}* ;
- une **position horizontale** *(x, y), 1<=x,y<=10* ;
- une **position verticale** *{sol, plafond, porté, nom d'objet}* ;
- un **poids** *{lourd, léger}*.

Le singe

Le singe, étant unique, n'a pas besoin d'être identifié par un nom. Par contre, comme pour les objets physiques, il faut pouvoir le localiser (contrainte 9). Le type de localisation est sensiblement le même, c'est-à-dire une position horizontale sous la forme d'un couple d'entiers compris entre 1 et 10, et une position verticale semblable à celle d'un objet physique à deux valeurs près : "plafond" (puisque le

singe ne peut pas être au plafond[6]) et "porté" (puisque cette valeur signifie "porté par le singe, et qu'il ne peut pas se porter lui-même).

Enfin, il est intéressant de donner dans la représentation du singe une information sur ce qu'il tient en main : rien s'il a les mains libres, ou un nom d'objet s'il tient cet objet. Strictement parlant, cette information n'est pas nécessaire puisqu'on peut la déduire de la valeur de position verticale des objets physiques (si un objet physique a "porté" pour valeur de position verticale, le singe tient cet objet ; si aucun n'a cette valeur de position verticale, le singe ne tient rien). Toutefois, elle permet d'examiner l'état du singe indépendamment de celui des objets physiques, ce qui peut être intéressant dans la représentation des actions du singe et de son but (objet porté par le singe = objet cible). Cette information introduit donc une redondance qui devra être contrôlée (invariants, preuves de consistance,...).

En résumé, nous pouvons associer au singe les attributs suivants :

- une **position horizontale** *(x, y), 1<=x,y<=10* ;
- une **position verticale** *{sol, nom d'objet}* ;
- une information sur l'**objet porté** *{rien, nom d'objet}*.

2.3. MODELISATION EN *ALBERT*

La modélisation du problème du singe que nous allons présenter ici est directement inspirée de l'exemple proposé par E. Dubois [DUBOIS93].

2.3.1. Vocabulaire d'ALBERT

Pour disposer d'une terminologie claire et non ambiguë, nous allons reprendre les termes définis dans [GEORGEFF87] relatifs aux problèmes de planification (voir 1.4.), et mettre en évidence les similitudes et les différences de ces termes avec ceux du vocabulaire d'ALBERT.

En ALBERT, la notion d'**agent** est plus large que dans [GEORGEFF87] : un agent ALBERT peut être passif, c'est-à-dire subir les actions perpétrées par d'autres agents, actifs ceux-ci. Un agent ALBERT est donc un objet avec un état, lequel peut être modifié par une action.

La notion d'**état** en ALBERT se rapporte plus directement à un agent que celle de [GEORGEFF87], qui parle plus généralement d' "état du monde". L'état d'un agent est une vision de celui-ci à un moment donné.

La notion de **comportement d'état** en ALBERT désigne les règles (invariants, contraintes temporelles) qui régissent les états ou les suites d'états possibles d'un agent plutôt que ces suites d'états elles-mêmes. En ALBERT, une suite d'états possibles d'un agent est appelée la **trace** d'un agent. Une séquence de changements d'état (décrits par l'occurrence d'une ou plusieurs actions simultanées) possibles d'un agent est une **histoire** de cet agent. Une séquence alternée d'états et de changements d'état d'un agent est une **vie** de cet agent.

Enfin, la notion d'**action** en ALBERT recouvre tout événement produit, qu'il ait ou non une influence sur l'état des agents.

---

[6]Le singe est soumis à la loi de la gravitation, et on suppose qu'il ne dispose pas de prise au plafond pour s'accrocher.

2.3.2. Première approche

Dans cette première approche d'analyse des besoins du problème du singe, nous considérons l'agent "chambre" comme un agent individuel dont les composantes sont le singe et l'ensemble des objets physiques. "Chambre" n'est donc pas un agent terminal au sens précisé dans [DUBOIS93], puisqu'on peut le décomposer en agents "singe" (individu) et "objet" (population). A ce niveau de raffinement, nous considérerons le singe et l'ensemble des objets physiques comme des attributs de l'agent "chambre". Dans la deuxième approche du problème, l'agent "chambre" sera vu comme un agent complexe, composé de deux agents terminaux (figure 2.1.).

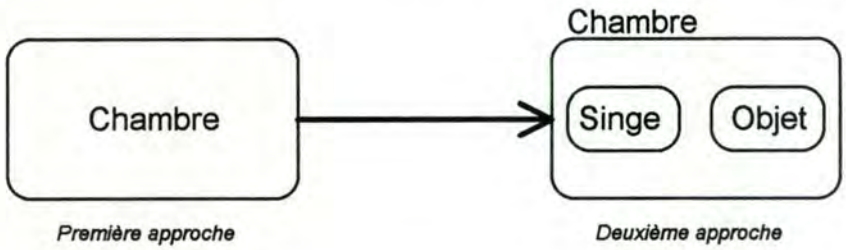

Figure 2.1. Transformation entre les approches ALBERT

Avant de proposer la déclaration associée à l'agent "chambre", déclarons d'abord les types complexes dont nous avons besoin[7] :

COMPLEX TYPE DECLARATIONS :

- **NOM-OBJET** = {"bananes", "lit", "couverture", "échelle"}
- **POS-HOR** = CP (abs : **COOR**, ord : **COOR**)
- **COOR** = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- **POS-VER-OBJET** = Union ({"sol", "plafond", "porté"}, **NOM-OBJET**)
- **POIDS-OBJET** = {"lourd", "léger"}
- **OBJET** = CP (nom : **NOM-OBJET**, pos-hor : **POS-HOR**, pos-ver : **POS-VER-OBJET**, poids : **POIDS-OBJET**)
- **POS-VER-SINGE** = Union ({"sol"}, **NOM-OBJET**)
- **OBJET-PORTE-SINGE** = Union ({"rien"}, **NOM-OBJET**)
- **SINGE** = CP (pos-hor : **POS-HOR**, pos-ver : **POS-VER-SINGE**, objet-porté : **OBJET-PORTE-SINGE**)

Cette définition de types complexes a pour but essentiel de typer le singe d'une part, et les composants de l'ensemble des objets physiques (à savoir les objets physiques eux-mêmes) d'autre part. Grâce à ce typage, nous pouvons élaborer le diagramme de déclaration de l'agent "chambre" (figure 2.2.).

[7]La syntaxe de définition de ces types complexes est encore laissée à l'appréciation du spécifieur en ALBERT. Nous adopterons un format libre inspiré de GLIDER [DUBOIS91].

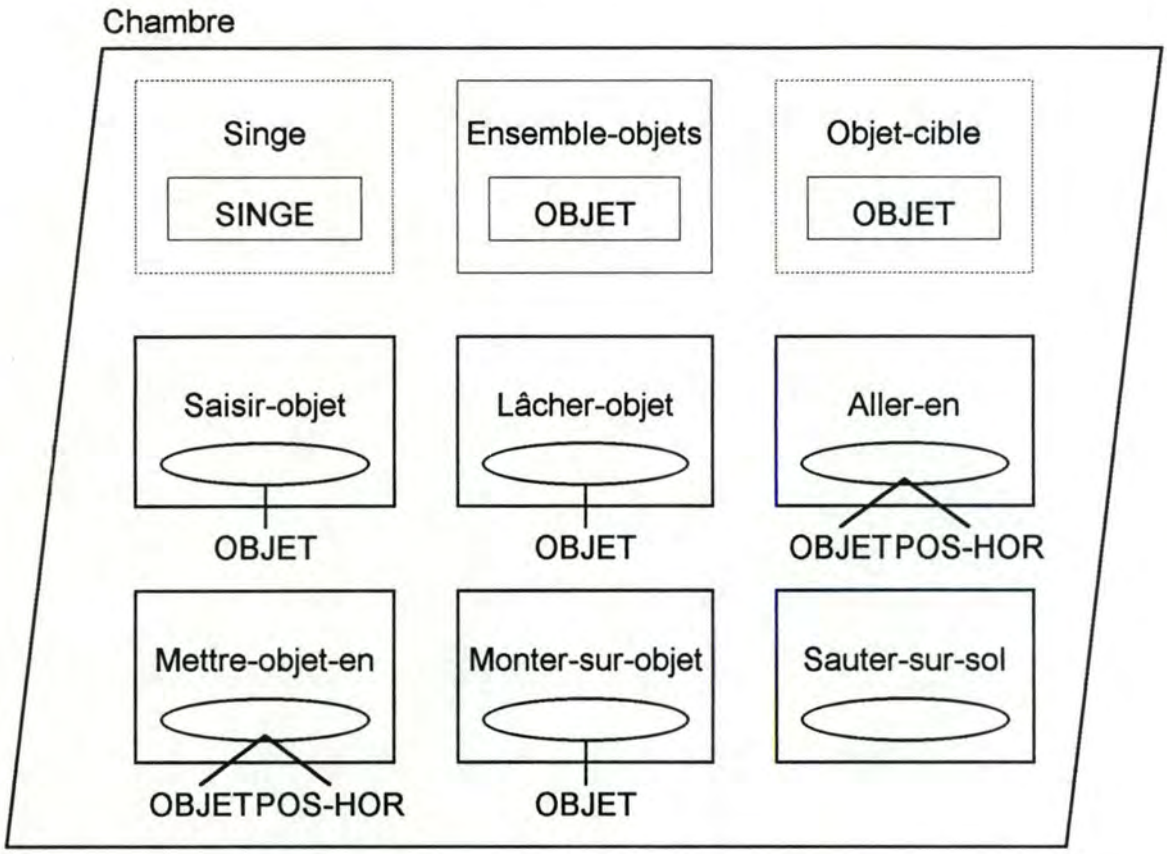

Figure 2.2. Diagramme de déclaration de l'agent Chambre

Cette déclaration indique que l'agent "chambre" :

- a trois attributs : un singe, individu de type SINGE, un ensemble d'objets, population d'objets de type OBJET , et un objet-cible, individu de type OBJET
- peut changer d'état suite à la survenance d'une des actions suivantes :
    - Saisir-objet (avec un argument de type OBJET)
    - Lâcher-objet (avec un argument de type OBJET)
    - Aller-en (avec un argument de type OBJET et un argument de type POS-HOR)
    - Mettre-objet-en (avec un argument de type OBJET et un argument de type POS-HOR)
    - Monter-sur-objet (avec un argument de type OBJET)
    - Sauter-sur-sol (sans argument)

Notons que l'agent "chambre" décrit entièrement les besoins du problème du singe et n'a aucune interaction avec le monde extérieur (d'où l'absence de flèches indiquant une visibilité de l'agent "chambre" par des agents extérieurs).

La figure 2.3 (page suivante) présente le diagramme de contraintes associé à l'agent "chambre".

Chambre

State Behaviour

In(Ensemble-objets,o) ∨ (o.nom = n) ⇒ □ (o.nom = n)
In(Ensemble-objets,o) ∨ (o.poids = p) ⇒ □ (o.poids = p)

In(Ensemble-objets,Objet-cible)
Objet-cible.poids = "léger"
(Objet-cible = o) ⇒ □ (Objet-cible = o)

◊ (Singe.objet-porté = Objet-cible.nom)

In(Ensemble-objets,o) ⇒ o.nom <> o.pos-ver

In({bananes,lit,couverture,échelle},nom1) ⇒ ∃o (In(Ensemble-objets,o) ∧ (o.nom = nom1))
In(Ensemble-objets,o1) ∧ In(Ensemble-objets,o2) ∧ (o1 <> o2) ⇒(o1.nom <> o2.nom)
In(Ensemble-objets,o1) ∧ In(Ensemble-objets,o2) ∧ (o1 <> o2) ∧ (o2.nom = o1.pos-ver) ⇒
(o1.pos-hor = o2.pos-hor)

(Singe.objet-porté <> "rien") ⇒ ∃o (In(Ensemble-objets,o) ∧ (o.nom = Singe.objet-porté) ∧
(o.pos-ver = porté))

Effects of Actions

Saisir-objet(o) : (o.pos-ver = "porté") ∧ (Singe.objet-porté = o.nom)
Lâcher-objet(o) : (o.pos-ver = "sol") ∧ (Singe.objet-porté = "rien")
Aller-en(xy) : (Singe.pos-hor = xy) ∧ ((∀o (In(Ensemble-objets,o) ∧ (Singe.objet-porté = o.nom))) ⇒
(o.pos-hor = xy))
Mettre-objet-en(o,xy) : (Singe.pos-hor = xy) ∧ (Singe.objet-porté = "rien") ∧ (o.pos-hor = xy) ∧
(o.pos-ver = "sol")
Monter-sur-objet(o) : Singe.pos-ver = o.nom
Sauter-sur-sol : Singe.pos-ver = "sol"

Commitments

Agent Responsibilities

F(Saisir-objet(o) / ¬ In(Ensemble-objets,o))
F(Saisir-objet(o) / Singe.objet-porté <> "rien")
F(Saisir-objet(o) / o.poids = "lourd")
F(Saisir-objet(o) / Singe.pos-hor <> o.pos-hor)
F(Saisir-objet(o) / ∃o2 (In(Ensemble-objets,o2) ∧ (o2.pos-ver = o1.nom)))
F(Saisir-objet(o) / (o.pos-ver <> "plafond") ∧ (Singe.pos-ver <> "sol"))
F(Saisir-objet(o) / (o.pos-ver = "plafond") ∧ (Singe.pos-ver <> "échelle"))

F(Lâcher-objet(o) / ¬ In(Ensemble-objets,o))
F(Lâcher-objet(o) / Singe.objet-porté <> o.nom)

F(Aller-en (xy) / Singe.pos-ver <> "sol")

F(Mettre-objet-en(o,xy) / ¬ In(Ensemble-objets,o))
F(Mettre-objet-en(o,xy) / Singe.objet-porté <> o.nom)
F(Mettre-objet-en(o,xy) / Singe.pos-ver <> "sol")

F(Monter-sur-objet(o) / ¬ In(Ensemble-objets,o))
F(Monter-sur-objet(o) / Singe.pos-ver <> "sol")
F(Monter-sur-objet(o) / o.pos-ver <> "sol")
F(Monter-sur-objet(o) / Singe.pos-hor <> o.pos-hor)
F(Monter-sur-objet(o) / Singe.objet-porté <> "rien")

F(Sauter-sur-sol / Singe.pos-ver = "sol")

Figure 2.3. Diagramme de contraintes associé à l'agent Chambre

Le diagramme de contraintes est divisé en quatre parties que nous détaillons ci-dessous.

Comportement d'état ("state behaviour")

Cette partie reprend les contraintes du système. Celles-ci sont soit des invariants (contraintes vraies pour tout état de l'agent "chambre"), soit des contraintes temporelles (contraintes sur l'évolution du système). Nous les avons scindées en six groupes.

Le premier groupe exprime la constance des attributs "nom" et "poids" des objets physiques, c'est-à-dire qu'un objet ne change jamais de nom ni de poids au cours de l'évolution du système.

Le deuxième groupe est relatif au concept d'objet-cible. Il exprime que l'objet cible est un des objets présents dans l'ensemble des objets physiques, et d'autre part qu'il ne change pas au cours de l'évolution du système.

Le troisième groupe est composé d'une seule contrainte qui exprime le but du problème. Cette contrainte indique que le singe portera l'objet-cible dans ses bras tôt ou tard, dans un état futur de l'agent "chambre".

Les quatrième, cinquième et sixième groupes contiennent les invariants relatifs aux états de l'agent "chambre". Ces invariants étant assez nombreux, nous nous sommes contentés d'en présenter quelques exemples significatifs.

Le quatrième groupe contient un exemple d'invariant relatif à un objet physique pris isolément, c'est-à-dire indépendamment des autres objets ou du singe. Cet invariant exprime le fait qu'un objet physique ne peut être posé sur lui-même (sa valeur de position verticale ne peut pas être le nom qui le désigne lui-même).

Le cinquième groupe exprime des invariants relatifs à l'ensemble des objets physiques pris comme un tout et indépendant du singe. Les deux premiers invariants indiquent que chaque objet physique (désigné par son nom) a une et une seule représentation dans l'ensemble des objets physiques. Le troisième invariant indique que si un objet physique est posé sur un autre objet, alors ces objets sont à la même position horizontale.

Enfin, le sixième groupe exprime des invariants relatifs à l'agent "chambre" pris de manière globale. L'exemple présenté indique que si le singe porte quelque chose, alors ce "quelque chose" est un objet qui existe dans l'ensemble des objets physiques et dont la valeur de position verticale est "porté" (cet invariant est celui qui contrôle la redondance que nous évoquions à la fin du point 2.2.4.).

Effets des actions ("effects of actions")

Dans cette partie, le diagramme de contraintes décrit les effets que l'exécution d'une action a sur l'agent "chambre". Ces effets correspondent aux postconditions décrites dans le point 2.2.3. et leur compréhension est immédiate.

Remarquons toutefois le cas particulier de l'action **aller-en** qui a un effet de bord : si le singe porte un objet et qu'il exécute cette action, l'objet se déplace avec lui. Nous décrivons cela en indiquant que l'effet de l'action **aller-en (xy)** sur tout objet dont la valeur de position verticale est "porté", est que la valeur de position horizontale de cet objet devient xy.

Règles de causalité ("commitments")

Cette partie décrit les liens de causalité entre des occurrences d'actions du système (obligation que telle action se produise suite à telle autre). Dans ce problème, nous n'en rencontrons pas : aucune action du singe ne conditionne obligatoirement l'exécution d'une autre.

Responsabilité des agents ("agent responsibilities")

Cette partie définit le rôle des agents en indiquant les conditions sous lesquelles il doivent ou ne peuvent pas exécuter certaines actions. Dans notre cas, toutes les actions sont permises si et seulement si leurs préconditions sont respectées. Cela revient à dire qu'elles sont interdites lorsque ces préconditions ne sont pas respectées. Les contraintes exprimées dans cette partie sont donc de la forme *F(action / ¬ précondition)*, ce qui veut dire que l'action ne peut être exécutée si la précondition n'est pas respectée.

L'absence de clauses *O* et *X* à ce niveause justifie par le fait qu'aucune action n'est jamais obligatoire dans notre problème.

2.3.3. Deuxième approche

Dans la première approche d'analyse des besoins que nous venons de présenter, nous avons considéré un seul agent dans le problème, l'agent "chambre". Il est clair que cette approche n'est pas terminale, en ce sens qu'elle ne met pas en évidence les agents réels du problème qui sont le singe, agent actif, et les objets physiques, agents passifs. Dans l'approche qui suit, nous allons transformer cette première approche en une spécification terminale de ces différents agents.

La chambre

L'agent "chambre" est donc dorénavant décrit comme un agent complexe, composé d'une instance d'agent "singe" et de plusieurs instances d'agent "objet" (figure 2.4.).

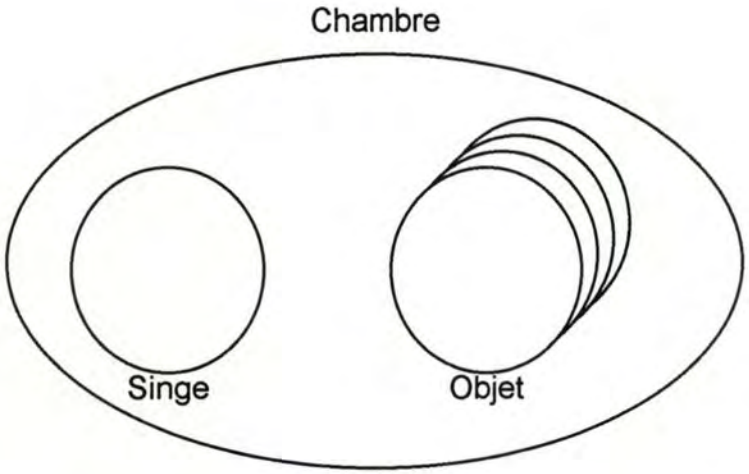

Figure 2.4. Diagramme de déclaration de l'agent complexe Chambre

Remarquons que l'objet cible n'est pas considéré comme un agent dans ce diagramme : il ne s'agit en effet pas d'un agent à part entière, puisque son comportement n'est pas différent de celui des objets physiques dont il fait partie. Sa spécificité en tant que cible du problème n'est connue que du singe : il sera donc décrit comme une composante de l'agent "singe".

Avant de décrire les agents "singe" et "objet", déclarons d'abord les types complexes dont nous avons besoin (il s'agit en fait d'un sous-ensemble des types déclarés dans la première approche).

COMPLEX TYPE DECLARATIONS :

- **NOM-OBJ** = {"bananes", "lit", "couverture", "échelle"}
- **PH** = CP (abs : **COOR**, ord : **COOR**)
- **COOR** = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- **PV-OBJ** = Union ({"sol", "plafond", "porté"}, **NOM-OBJ**)
- **POIDS-OBJ** = {"lourd", "léger"}
- **PV-SINGE** = Union ({"sol"}, **NOM-OBJ**)
- **OBJ-PORTE** = Union ({"rien"}, **NOM-OBJ**)

Le singe

Le singe est l'agent actif du problème : il a l'initiative des actions.

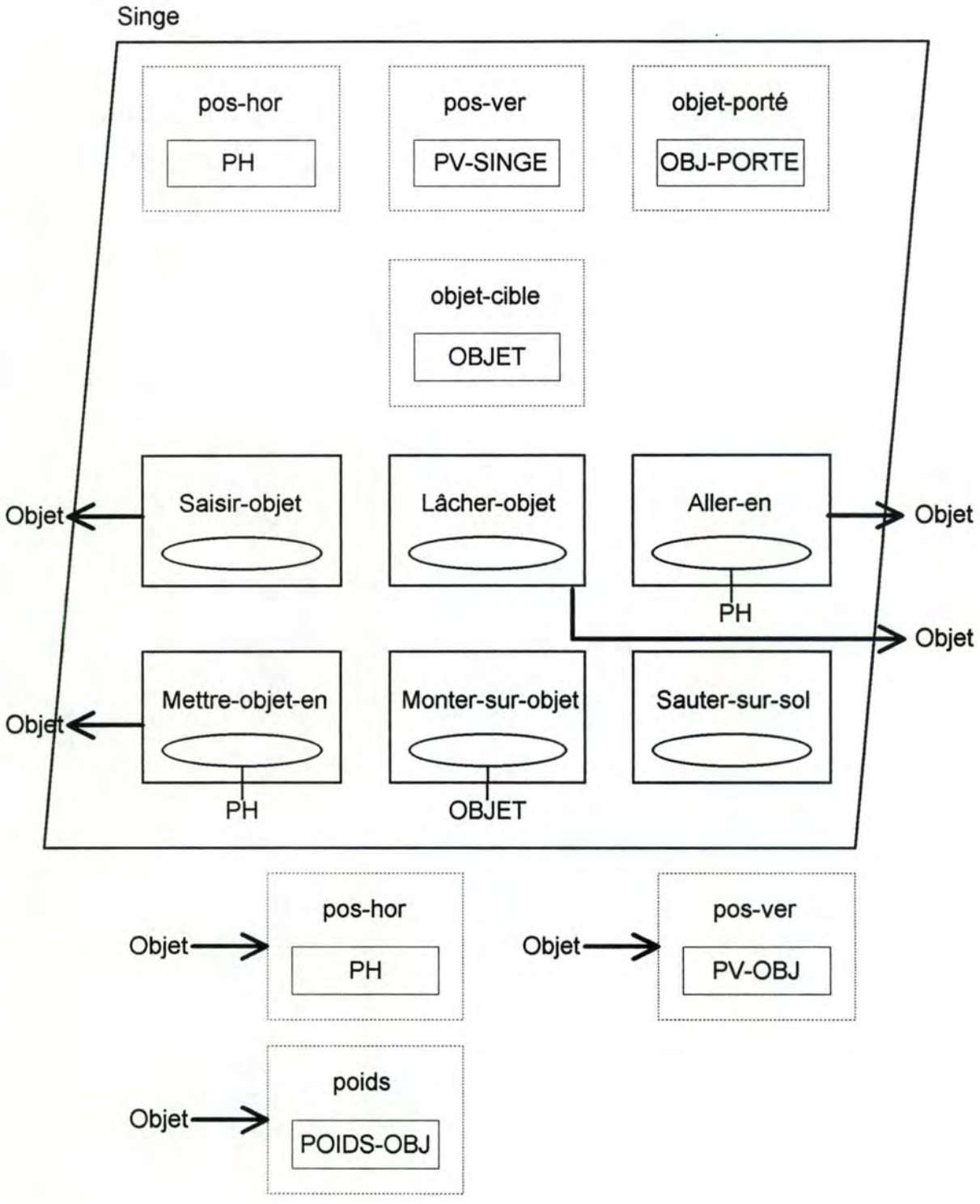

Figure 2.5. Diagramme de déclaration associé à l'agent Singe

Dans ce diagramme, le type OBJET correspond à l'agent "objet" que nous définissons par la suite.

Le singe est caractérisé par quatre composantes :

- sa position horizontale
- sa position verticale
- l'objet qu'il porte
- l'objet qu'il a pour cible

Il peut exécuter les six actions que nous avons décrites sur le système. Il doit également avoir une perception de trois composantes de toute instance d'agent "objet", à savoir la position horizontale, la position verticale et le poids de cet objet, afin de pouvoir exécuter les actions qui ont un effet sur l'objet.

Singe

State Behaviour

Objet-cible.poids = "léger"
(Objet-cible = o) ⇒ □ (Objet-cible = o)

◊ (Singe.objet-porté = Objet-cible.nom)

Effects of Actions

Saisir-objet_{o} : objet-porté = o.nom
Lâcher-objet_{o} :objet-porté = "rien"
Aller-en(xy) : pos-hor = xy
Mettre-objet-en(xy)_{o} : (pos-hor = xy) ∧ (objet-porté = "rien")
Monter-sur-objet(o) : pos-ver = o.nom
Sauter-sur-sol : pos-ver = "sol"

Commitments

Agent Responsibilities

F(Saisir-objet_{o} / objet-porté <> "rien")
F(Saisir-objet_{o} / o.poids = "lourd")
F(Saisir-objet_{o} / pos-hor <> o.pos-hor)
F(Saisir-objet_{o} / ∃objet (objet.pos-ver = o.nom)))
F(Saisir-objet_{o} / (o.pos-ver <> "plafond") ∧ (pos-ver <> "sol"))
F(Saisir-objet_{o} / (o.pos-ver = "plafond") ∧ (pos-ver <> "échelle"))

F(Lâcher-objet_{o} / objet-porté <> o.nom)

F(Aller-en (xy)_{o} / pos-ver <> "sol")

F(Mettre-objet-en(xy)_{o} / objet-porté <> o.nom)
F(Mettre-objet-en(xy)_{o} / pos-ver <> "sol")

F(Monter-sur-objet(o) / pos-ver <> "sol")
F(Monter-sur-objet(o) / o.pos-ver <> "sol")
F(Monter-sur-objet(o) / pos-hor <> o.pos-hor)
F(Monter-sur-objet(o) / objet-porté <> "rien")

F(Sauter-sur-sol / pos-ver = "sol")

Figure 2.6. Diagramme de contraintes associé à l'agent Singe

Nous commenterons plus loin les correspondances entre le diagramme de contraintes de l'agent "chambre" dans la première approche, et ceux des agents "singe" et "objet" dans la deuxième approche.

Les objets physiques

Les objets physiques sont des agents passifs.

Le diagramme de déclaration associé à l'agent "objet" (figure 2.7.) indique qu'un objet :

- est composé de quatre attributs : un nom, une position horizontale, une position verticale et un poids. Les trois derniers sont visibles de l'extérieur par le singe.
- peut changer d'état suite à l'occurrence de quatre actions, toutes à l'initiative du singe : Saisir-objet, Lâcher-objet, Aller-en, Mettre-objet-en.

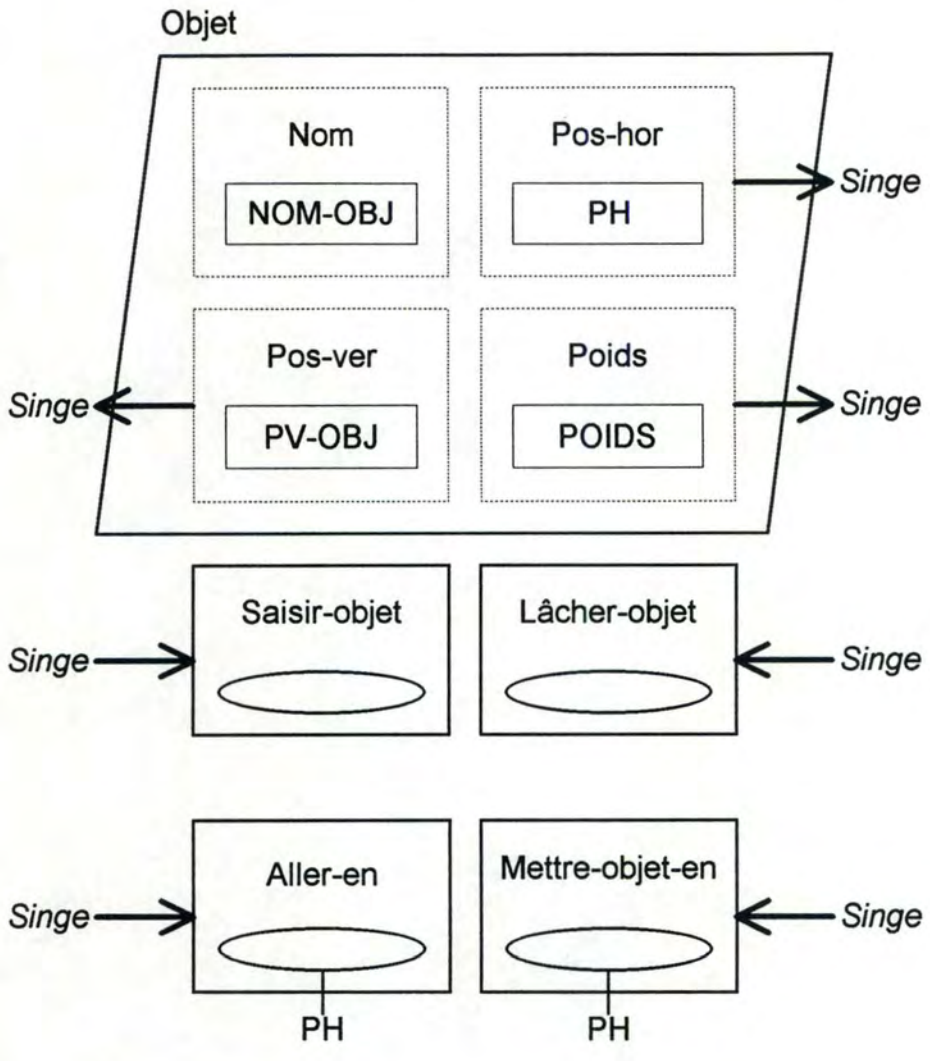

Figure 2.7. Diagramme de déclaration de l'agent Objet

La figure 2.8. présente le diagramme de contraintes associé à l'agent "objet".

Objet

STATE BEHAVIOUR

nom = n $\Rightarrow$ $\Box$ *(nom = n)*
poids = p $\Rightarrow$ $\Box$ *(poids = p)*

nom <> pos-ver

EFFECTS OF ACTIONS

singe.Saisir-objet : pos-ver = "porté"
singe.Lâcher-objet : pos-ver = "sol"
singe.Aller-en(xy) : pos-hor = xy
singe.Mettre-objet-en(xy) : pos-hor = xy ∧ pos-ver = "sol"

COMMITMENTS

AGENT RESPONSIBILITIES

O(singe.Saisir-objet / true)
O(singe.Lâcher-objet / true)
X(singe.Aller-en (xy) / pos-ver = "porté"))
O(singe.Mettre-objet-en(xy) / true)

O(pos-hor_{singe} / true}
O(pos-ver_{singe} / true}
O(poids_{singe} / true}

Figure 2.8. Diagramme de contraintes associé à l'agent Objet

Dans ce diagramme de contraintes, les responsabilités de l'agent "objet" expriment deux choses :

- d'une part, que lorsque le singe effectue les actions qui ont un effet sur un objet (Saisir-objet, Lâcher-objet, Aller-en, Mettre-objet-en), l'objet concerné est obligé de subir l'action (clause O). La clause X associée à l'action "Aller-en" exprime que l'objet ne subit l'action que si sa position verticale est "porté", et qu'elle n'a aucun effet sur lui s'il n'a pas cette valeur de position verticale (ce qui est une façon de spécifier l'effet de bord de la fonction "Aller-en") ;
- d'autre part, que l'objet doit toujours offrir une perception de ses attributs de position et de poids au singe (puisque celui-ci doit les connaître pour effectuer une action).

2.3.4. Transformations opérées entre les deux approches

L'agent "chambre" a été décomposé en deux types d'agents, le singe et les objets physiques. Chacun d'eux est composé d'attributs de position, de poids, etc. Le raffinement de la spécification permet de préciser la visibilité de ces attributs par les autres agents.

Le singe

En tant qu'agent actif, l'agent "singe" est notamment défini par un ensemble d'actions qu'il peut exécuter. Ce sont les actions déjà identifiées pour l'agent "chambre".

Au niveau du **comportement d'état**, on ne retrouve plus ici que les contraintes relatives au singe lui-même (et donc à l'objet cible puisque celui-ci est modélisé comme une composante de l'agent "singe"). Remarquons que le singe n'a pas d'invariant propre.

Au niveau des **effets des actions**, on retrouve simplement les effets de ces actions sur le singe lui-même, indépendamment des conséquences sur l'état des objets.

Au niveau des **responsabilités**, on retrouve l'expression de toutes les préconditions décrites pour l'agent "chambre"[8] (pour rappel, sous la forme négative *F(action / ¬ précondition)*), qui impliquent le singe.

Les objets

En tant qu'agent passif, un objet est défini par les actions qu'il subit et qui ont un effet sur lui. Celles-ci sont un sous-ensemble des actions définies pour l'agent "chambre" et correspondent à une action "active" décrite pour le singe.

Au niveau du **comportement d'état**, seules les contraintes relatives à chaque objet (indépendamment des autres et du singe) sont reprises. Les autres contraintes identifiées pour la chambre qui mettent en relation différents agents "objet" et "singe" disparaissent à ce niveau de raffinement. Elles restent toutefois dérivables du reste de la spécification (notamment de la description des effets des actions).

Au niveau des **effets des actions**, on retrouve les effets sur l'objet, indépendamment des autres effets sur le singe.

Au niveau des **responsabilités**, on exprime la responsabilité des agents "objet" face aux actions perpétrées à l'initiative du singe, ainsi que la perception que ces agents doivent offrir d'eux-mêmes à l'agent "singe".

---

[8] A l'exception de celles demandant qu'un argument "objet" appartienne à l'ensemble des objets. Cet ensemble est devenu une classe d'agents, et il suffit de dire que le type de l'argument correspond à un identifiant de cet agent;

CHAPITRE 3
KIF

REMARQUE PRELIMINAIRE: Ce chapitre a pour objectif de présenter au lecteur le langage KIF de façon synthétique, afin que d'une part, il puisse s'en faire une idée globale, et d'autre part il soit préparé à suivre l'exposé du chapitre 4, dans lequel nous présenterons la spécification d'un problème pratique en KIF (le problème du singe exposé au chapitre 2). Dans ce chapitre-ci, nous nous inspirerons du manuel de référence de KIF version 3.0. [GENESERETH92], sans toutefois nous substituer à lui. Ainsi, nous laisserons de côté certains concepts du langage qui sont de peu d'utilité pour sa compréhension intuitive, ainsi que des aspects sans intérêt pour la mise en pratique du chapitre qui suit. Le lecteur qui désire appréhender en détail l'entièreté des concepts de KIF est invité à lire attentivement le manuel de référence, beaucoup plus complet que ce qui va suivre, bien que, pensons-nous, la compréhension n'en soit pas aisée.

3.1. INTRODUCTION

KIF, abréviation de *Knowledge Interchange Format* (format d'échange de connaissances), est un langage conçu pour l'échange de connaissances entre programmes d'ordinateurs disparates. Supposons par exemple deux langages de programmation LP1 et LP2 et un programme P écrit en LP1. KIF est conçu de telle façon qu'il est possible de faire une traduction systématique de la représentation des connaissances contenue dans P, en une représentation en KIF. De plus, il est possible d'opérer une traduction systématique en sens inverse, de façon à obtenir une représentation équivalente en LP2, de telle sorte que cette représentation des connaissances en LP2 est sémantiquement équivalente à celle contenue dans P.

KIF n'est donc pas conçu comme un langage d'interaction avec les utilisateurs humains, ni comme un langage de représentation interne des connaissances au sein des programmes d'ordinateurs. Toutefois, il peut être utilisé dans ces deux optiques.

Les points suivants sont des propriétés essentielles de KIF :

1. le langage a une *sémantique déclarative* propre, c'est-à-dire qu'il est possible de comprendre le sens de ses expressions sans devoir faire appel à un interpréteur spécifique ;
2. le langage permet l'expression de n'importe quelle proposition logique dans le calcul des prédicats ;
3. le langage permet la représentation des *métaconnaissances* (les connaissances sur les connaissances).

Ces critères sont essentiels. De plus, KIF a les propriétés suivantes, d'importance moindre mais non négligeable :

- "traductibilité" : KIF donne des moyens pratiques de traduire des bases de connaissances déclaratives dans des langages de représentation des connaissances, et vice versa ;
- lisibilité : Il doit être possible d'utiliser KIF comme langage d'interaction avec l'être humain, pour décrire la sémantique de langages de représentation, pour

représenter des exemples de bases de connaissances , et dans notre cas, pour permettre la spécification abstraite d'un problème à résoudre par système expert ;
- "utilisabilité" comme langage de représentation (même si fondamentalement ce n'est pas l'optique dans laquelle KIF a été mis au point).

La syntaxe BNF de KIF est présentée en annexe.

3.2. CONCEPTS DE BASE

Il existe trois concepts importants en KIF : les *objets*, les *fonctions* et les *relations*.

3.2.1. Objets

En KIF, la notion d'objet est large. Il s'agit de tous les objets existants ou supposés exister dans le monde. Ces objets peuvent être concrets ou abstraits, primitifs ou composés d'autres objets, etc. L'ensemble de ces objets est appelé un *univers du discours*. Chaque utilisateur a sa propre vision d'un problème, donc son propre univers du discours. Toutefois, certains objets sont communs à tous ces univers. Il s'agit des objets suivants :

- les *mots* : les mots sont des objets, comme les objets qu'ils dénotent ;
- les *nombres* (y compris les nombres complexes) ;
- les *listes* finies d'objets ;
- les *ensembles* finis d'objets ;
- l'objet $\perp$ ("bottom"), un objet particulier qui est la valeur que prend une fonction quand son résultat est indéterminé.

Outre ces objets de base, l'utilisateur peut définir les objets qu'il veut.

3.2.2. Fonctions

Il y a deux façons en KIF de concevoir une fonction.

On peut d'abord voir une fonction comme un ensemble de listes d'objets dans lesquelles le dernier élément - le *résultat* - est déterminé par les éléments précédents - les *arguments*. Toute liste qui associe ainsi une série d'arguments à un résultat appartient à cette fonction[9].

L'autre façon de concevoir une fonction est bien sûr la vision classique, c'est-à-dire une correspondance entre des arguments et un résultat. Le rapprochement entre les deux visions est immédiat (et supporté par la sémantique de KIF).

Exemple : si l'on considère la fonction "diviser", qui renvoie le quotient de deux nombres, les listes [1, 1, 1] et [8, 4, 2] appartiennent à l'ensemble qui définit la fonction, tandis que [1, 2, 3] ne lui appartient pas.

---

[9]On pose toutefois que les listes se terminant par $\perp$ qui correspondent au cas où la fonction n'est pas applicable aux arguments donnés, ne font pas partie de l'ensemble qui définit la fonction.

3.2.3. Relations

Comme dans le cas des fonctions, il y a deux manières de concevoir une relation.

Une relation est d'abord un ensemble de listes d'objets (sans contrainte sur le dernier élément). Les éléments d'une liste qui appartient à la relation *vérifient* cette relation.

On peut aussi voir une relation comme l'expression d'un prédicat auquel on associe une liste d'arguments. Si cette liste appartient à l'ensemble qui définit la relation, la valeur de vérité du prédicat est *vrai*, sinon cette valeur est *faux*.

Exemple: considérons la relation "est-une-progression", c'est-à-dire que les éléments des listes qui vérifient cette relation forment une progression soit arithmétique, soit géométrique. Il est clair que la liste [2, 4, 6, 8] appartient à l'ensemble définissant cette relation, alors que [7, 11, 24] ne lui appartient pas. Pareillement, on peut voir que le prédicat "est-une-progression ([2, 4, 6, 8])" a une valeur de vérité *vrai* et que "est-une-progression ([7, 11, 24])" a une valeur de vérité *faux*.

3.2.4. Différence entre fonction et relation

La définition d'une fonction et d'une relation sous forme d'ensemble de listes montre qu'une fonction est une relation particulière, dans laquelle le dernier élément de chaque liste dépend des précédents, c'est-à-dire que dans une fonction, il ne peut y avoir deux listes qui soient identiques au dernier élément près.

Cette différence permet de définir l'une et l'autre de deux manières différentes :

- une fonction peut se définir en donnant l'expression du terme résultat par rapport aux termes arguments. Dès lors, en associant à chaque suite d'arguments applicables un résultat par l'expression donnée de ce terme, on obtient une liste qui appartient à l'ensemble qui définit la fonction ;
- une relation peut se définir en donnant l'expression d'une proposition logique que vérifient ses arguments. Toutes les listes d'arguments qui vérifient cette proposition logique appartiennent à l'ensemble qui définit la relation.

Nous verrons plus loin ces définitions de façon formelle.

3.3. EXPRESSIONS

Il y a quatre expressions légales en KIF : les *termes*, les *propositions*, les *définitions* et les *règles*.

3.3.1. Termes

Un terme est toute façon de dénoter un objet dans le monde décrit (c'est-à-dire aussi bien les objets de l'univers du discours que les fonctions et les relations). Il y a dix types de termes : les *variables individuelles*, les *constantes d'objet*, les *constantes de fonction* et les *constantes de relation* sont les types de base ; les *termes de fonction*, les *termes de liste*, les *termes d'ensemble*, les *termes quotés*, les *termes logiques* et les *termes quantifiés* sont les types élaborés.

Variables individuelles et variables de séquence

Une **variable individuelle** (*<indvar>*) est un mot dont le premier caractère est ?, par exemple *?var*. Elle sert à quantifier l'expression d'un objet individuel.

Une **variable de séquence** (*<seqvar>*) est un mot dont le premier caractère est @, par exemple *@seq*. Elle sert à quantifier l'expression d'une suite d'objets.

Il est important de différencier ces variables au sens KIF du terme des variables au sens traditionnel des langages de programmation. Les variables individuelles et les variables de séquence n'ont pas de signification hors de la phrase KIF dans laquelle elles sont présentes. La meilleure façon de le comprendre est de prendre une expression mathématique quantifiée de la forme :

$\forall$ x, y :((x,y $\in$ **R** => x$\leq$y ou y$\leq$x)

Cette expression peut être écrite en KIF de la façon suivante :

```
((FORALL ?x ?y)
   (=> (real-number ?x) (real-number ?y)
      (OR (=< ?x ?y) (=< ?y ?x))))
```

(Les relations *real-number* et =< correspondent à leur signification intuitive et sont définies dans [GENESERETH92], chapitre 5).

Les variables individuelles *?x* et *?y* servent à quantifier la proposition ci-dessus, et n'ont de portée que dans cette proposition. Toute autre phrase KIF peut contenir *?x* ou *?y* sans qu'il y ait le moindre lien entre ces variables et celles de la proposition ci-dessus.

Une variable de séquence n'est pas un terme à proprement parler, puisqu'elle ne dénote pas un objet mais une suite d'objets.

Constantes

Une **constante d'objet** (*<objconst>*) est un mot qui dénote un objet particulier de l'univers du discours.

Une **constante de fonction** (*<funconst>*) est un mot qui dénote une fonction particulière.

Une **constante de relation** (*<relconst>*) est un mot qui dénote une relation particulière.

On verra que les constantes sont associées à l'objet qu'elle dénotent au moyen d'une définition KIF.

Termes élaborés

Un **terme de fonction** (*<funterm>*) est un terme composé d'une constante de fonction et d'une série de termes arguments. Il dénote le résultat de l'application de la fonction aux termes arguments. Exemple : *(diviser 8 4)* est le terme de fonction résultant de l'application de *diviser* aux objets *8* et *4*[10].

Un **terme de liste** (*<listterm>*) est un terme composé de l'opérateur *LISTOF* et d'une suite (vide ou non) de termes, éventuellement terminée par une variable de séquence. Il dénote une liste - c'est-à-dire une séquence ordonnée - d'objets. Exemples : *(LISTOF 1 2 3 4)* est un terme qui dénote la liste des quatre premiers

[10]Remarquons que les nombres sont des constantes d'objet, qu'il n'est pas nécessaire de définir puisqu'ils sont des objets de base.

nombres naturels strictement positifs ; *(LISTOF 5 @nombres)* dénote une liste dont le premier élément est le nombre 5, les suivants une suite d'objets quelconques.

Un **terme d'ensemble** (*<setterm>*) est un terme composé de l'opérateur *SETOF* et d'une suite (vide ou non) de termes, éventuellement terminée par une variable de séquence. Il dénote un ensemble (au sens mathématique du mot) d'objets. Exemples : *(SETOF 1 2 3 4)* est un terme qui dénote l'ensemble des nombres entiers naturels strictement positifs plus petits ou égaux à 4 ; *(SETOF 5 @nombres)* dénote un ensemble contenant le nombre 5 et un certain nombre d'objets quelconques.

Un **terme quoté** (*<quoterm>*) est un terme composé de l'opérateur *quote* (ou ') et d'une expression quelconque. Il dénote un objet qui est l'expression en question. Par exemple, si le terme *soleil* dénote l'objet appelé "soleil", le terme *'soleil* dénote quant à lui le mot "soleil" et non l'objet désigné par ce mot (rappelons que les mots sont des objets de base de l'univers du discours).

Un **terme logique** ou **terme conditionnel** (*<logterm>*) est un terme dont l'objet qu'il dénote dépend de la valeur de vérité d'une certaine proposition (nous parlons des propositions KIF dans le point suivant). Il y en a de deux sortes :

- le *terme conditionnel simple* de la forme *(IF <proposition> <terme> [<terme>])* où l'objet dénoté par le terme conditionnel est celui dénoté par le premier terme mentionné si la proposition est vraie, celui dénoté par le second terme mentionné (ou $\perp$ s'il n'y a pas de second terme) si la proposition est fausse. Exemple de terme conditionnel simple : *(IF (< ?x 0) ('négatif) ('positif))* où le terme conditionnel dénote le mot "négatif" si le terme dénoté par ?x est un nombre plus petit que zéro, ou dénote le mot "positif" dans tous les autres cas. REMARQUE: le terme conditionnel simple sera très utilisé au chapitre 4, lors de la mise en pratique.
- le *terme conditionnel multiple* de la forme *(COND (<proposition> <terme>) ... (<proposition> <terme>))* où l'objet dénoté par le terme conditionnel est celui dénoté par le premier terme dont la proposition associée est vraie, ou $\perp$ si aucune des propositions ne l'est.

Enfin, un **terme quantifié** (*<quantterm>*) est un terme qui dénote un objet quantifié. Il y en a de quatre types : *terme désignateur*, *terme formateur de fonction*, *terme formateur de relation*, et *terme formateur d'ensemble*. Nous n'expliquerons pas les trois premiers[11] (voir à ce sujet [GENESERETH92], chapitres 2 et 4), mais nous parlerons du *terme formateur d'ensemble*, de la forme *(SETOFALL <terme> <proposition>)*, qui dénote l'ensemble de tous les objets dénotés par le terme contenu dans sa description pour lesquels la proposition donnée est vraie. Exemple de terme formateur d'ensemble : *(SETOFALL (LISTOF ?x ?y) (AND (integer ?x) (integer ?y))* est l'ensemble de toutes les listes [x, y] ou x et y sont des entiers.

---

[11]En bref : un terme désignateur permet de dénoter un objet unique qui répond à une certaine condition (par exemple, élément minimum d'un ensemble de nombres), et les termes formateurs de fonction et de relation permettent de dénoter une fonction ou une relation sous leur forme ensembliste. Ils ne sont d'ailleurs pas indispensables, car ils sont dénotables par un terme formateur d'ensembles.

3.3.2. Propositions

Une proposition en KIF est une phrase logique ayant une valeur de vérité. Il y a six types de propositions: les *constantes logiques*, les *égalités*, les *inégalités*, les *propositions relationnelles*, les *propositions composées* et les *propositions quantifiées.*

Constantes logiques

Ce sont les constantes booléennes traditionnelles *true* et *false*.

Egalités

Ce sont les propositions de la forme *(= <terme> <terme>)*. La valeur de vérité d'une égalité est "vrai" si les deux termes de sa description dénotent le même objet.

Inégalités

Ce sont les propositions de la forme *(/= <terme> <terme>)*. La valeur de vérité d'une égalité est "vrai" si les deux termes de sa description dénotent deux objets différents.

Propositions relationnelles

Ce sont les propositions de la forme *(<relconst> <terme>* [<seqvar>])*. La valeur de vérité d'une proposition relationnelle est "vrai" si la suite d'arguments composée des objets dénotés par les termes et la variable de séquence (si elle est présente) , vérifie la relation dénotée par la constante de relation. En d'autres mots, cette valeur de vérité est "vrai" si la liste formée des termes et de la variable de séquence (si elle est présente) appartient à l'ensemble définissant la relation dénotée par la constante de relation[12].

Exemple: si on reprend la relation est-une-progression définie à la section 2, la proposition relationnelle (est-une-progression 2 4 ?x) est vraie si et seulement si la variable individuelle ?x dénote le nombre 6 ou le nombre 8. Elle est fausse dans tous les autres cas.

Propositions composées

Ce sont toutes les propositions formées à partir d'autres propositions et d'opérateurs logiques. Elles ont l'une des formes suivantes :

1. *(NOT <proposition>)*
2. *(AND <proposition>*)*
3. *(OR <proposition>*)*
4. *(=> <proposition>* <proposition>)*
5. *(<= <proposition> <proposition>*)*
6. *(<=> <proposition> <proposition>)*

---

[12]Puisqu'une fonction est un type de relation particulier, les auteurs de KIF ont envisagé la possibilité d'écrire une proposition relationnelle avec une constante de fonction à la place de la constante de relation. Cela nous semble une erreur, car une telle écriture peut aussi représenter un terme fonctionnel, et qu'il y a confusion entre ce terme et la proposition relationnelle, alors qu'un terme et une proposition sont deux concepts très différents.

Les valeurs de vérité des propositions composées se déduisent de celles des propositions qui les composent et de l'opérateur qui les lie (suivant la logique des prédicats du premier ordre). Notons que dans la proposition composée 4, le conséquent est la dernière proposition, toutes les autres étant les antécédents, et que la proposition composée 5 est l'expression inversée de la 4.

Exemple: la proposition composée *(OR (= x y) (/= x y))* est vraie quels que soient les objets dénotés par les constantes d'objet x et y.

Propositions quantifiées

Ce sont les propositions dans lesquelles intervient un quantificateur mathématique.

Une *proposition quantifiée universellement* est une proposition de la forme *(FORALL (<indvar>* [<seqvar>]) (<proposition>))*. Elle a "vrai" pour valeur de vérité si la proposition présente dans sa description est vraie quels que soient les objets dénotables[13] par les variables individuelles et la variable de séquence (si elle est présente) qui suivent le FORALL dans la description.

Exemple: (FORALL (?x ?y) (OR (= ?x ?y) (/= ?x ?y))) est vraie car pour tout couple de deux objets quelconques (représentés par les variables individuelles *?x* et *?y*, l'un est égal à l'autre ou l'un est différent de l'autre.

Remarque: En général, une proposition quantifiée universellement a la forme suivante : *(FORALL (<indvar>* [<seqvar>]) (=> (<proposition>* <proposition>)*, ce qui veut dire: "Pour tous les objets dénotables par la suite de variables derrière le *FORALL* et qui vérifient les propositions antécédentes de la proposition composée, alors la proposition conséquente est vérifiée". Exemple: *(FORALL (?x ?y) (=>(integer ?x) (integer ?y) (OR (=< ?x ?y) (=< ?y ?x))))*.

Une *proposition quantifiée existentiellement* est une proposition de la forme *(EXISTS (<indvar>* [<seqvar>]) (<proposition>))*. Elle a "vrai" pour valeur de vérité si la proposition présente dans sa description est vraie pour au moins une suite d'objets dénotables par les variables individuelles et la variable de séquence (si elle est présente) qui suivent le *EXISTS* dans la description.

Exemple: *(EXISTS (?x ?y) (= ?x ?y))* est vraie car il existe bien deux objets qui sont égaux entre eux (par exemple les nombres 4 et 4).

3.3.3. Définitions

Les définitions KIF servent à définir les concepts de base, c'est-à-dire les objets de l'univers du discours, les fonctions et les relations. Ces définitions peuvent être *complètes* ou *partielles* suivant que l'on connaît complètement l'objet à définir ou qu'on n'en connaît que certaines caractéristiques.

Définitions complètes

Ce sont les définitions qu'on donne d'un objet, d'une relation ou d'une fonction lorsque celui ou celle-ci est complètement connu. Il s'agit donc de l'idée intuitive qu'on peut avoir de la notion de définition.

Une **définition d'objet** associe à une constante d'objet un terme qui dénote l'objet à définir. Elle est de la forme *(DEFOBJECT <objconst> := <terme>)*. Par

[13]"dénotables" et non "dénotés". Cela veut dire que chaque terme présent dans cette suite peut représenter N'IMPORTE QUEL objet.

exemple, supposons qu'on ait défini les objets *lune*, *soleil*, *Vénus*. On peut définir l'objet *astres* comme étant l'ensemble composé de la lune, du soleil et de Vénus : *(DEFOBJECT astres := (SETOF lune soleil Vénus))*.

Généralement, les objets primitifs (lune, soleil) sont définis sous forme d'une liste de caractéristiques (par exemple, diamètre de l'astre, position par rapport à la Terre, distance par rapport à la Terre, etc.) donc sous la forme d'un terme de liste. Notons que, contrairement au concept d'objet du génie logiciel, l'objet KIF est constant et ne change pas d'état une fois défini.

Une **définition de fonction** associe à une constante de fonction et à une suite de variables dénotant les objets arguments de la fonction (variables individuelles suivies d'une variable de séquence optionnelle), un terme résultat qui est fonction des arguments. Elle est de la forme *(DEFFUNCTION <funconst> (<indvar>* [<seqvar>]):= <terme>)*. Par exemple, la fonction *moyenne*, qui associe à deux nombres entiers leur moyenne arithmétique, est définie comme suit :

```
(DEFFUNCTION moyenne (?x ?y) :=
    (IF (AND (integer ?x) (integer ?y)) (div (+ ?x ?y) 2)))
```

(on suppose + et *div* définies).

Une **définition de relation** associe à une constante de relation et à une suite de variables dénotant les objets arguments de la relation (variables individuelles suivies d'une variable de séquence optionnelle), une proposition KIF. Elle est de la forme *(DEFRELATION <relconst> (<indvar>* [<seqvar>]) := <proposition>)*. Les objets dénotés par les variables arguments et qui vérifient cette proposition forment une liste qui appartient à l'ensemble définissant la relation. Par exemple, la relation *est-une-progression-de-trois-éléments*, qui est vraie pour toute suite de trois objets qui sont des nombres en progression arithmétique ou géométrique, peut se définir comme suit :

```
(DEFRELATION est-une-progression-de-trois-éléments (?x ?y ?z) :=
    (AND (integer ?x) (integer ?y) (integer ?z)
        (OR (= (- ?y ?x) (- ?z ?y))
        (= (div ?y ?x) (div ?z ?y)))))
```

Toutes les listes de trois objets qui vérifient cette proposition vérifient bien la relation définie par celle-ci.

Définitions partielles

Dans un système de connaissances , il arrive que l'on ne dispose pas à un moment donné de toute l'information permettant de définir un objet, une fonction ou une relation. Les informations dont on dispose permettent seulement de restreindre le nombre d'objets dénotables par la constante à définir. KIF permet de résoudre ce problème grâce aux *définitions partielles*. Une définition partielle associe à chaque constante (d'objet, de fonction ou de relation) un ensemble de propositions KIF qui sont vérifiées par l'objet, la fonction ou la relation définie. On peut associer plusieurs définitions partielles à une même constante.

KIF distingue les *définitions partielles conservatrices* et les *définitions partielles non restreintes*. Nous nous contenterons de parler des secondes (le lecteur qui désire plus de détails sur les premières peut se référer à [GENESERETH92], chapitre 11).

Une **définition partielle d'objet** associe à une constante d'objet un ensemble de propositions KIF qui restreignent les dénotations possibles de l'objet. Elle est de la forme *(DEFOBJECT <objconst> <proposition>*)*. Par exemple, supposons que nous voulions définir le concept d'élément maximum d'un ensemble, mais que nous ne connaissions pas encore la relation d'ordre à appliquer sur cet ensemble. La seule chose que nous puissions dire de l'élément maximum de cet ensemble est qu'il appartient à l'ensemble en question:

(DEFOBJECT maximum
(member maximum ensemble-donné))

On suppose définis *ensemble-donné* et la relation *member* qui est vraie si son premier argument est un élément de l'ensemble donné en second argument.

Une **définition partielle de fonction** associe à une constante de fonction un ensemble de propositions qui restreignent les dénotations possibles de cette fonction (par exemple, en décrivant son domaine et son image).

Une **définition partielle de relation** associe à une constante de relation un ensemble de propositions qui restreignent les dénotations possibles de cette relation (il y a deux formes de définition partielle pour les relations. Voir [GENESERETH92] à ce sujet).

Dans les cas ou un problème est décomposé en une partie statique, indépendante de la configuration initiale, et une partie dynamique, dépendante d'une configuration initiale donnée, l'avantage des définitions partielles est que l'on peut inclure à la partie statique tout ce qu'on connaît d'un objet, d'une fonction ou d'une relation, même si on n'en connaît pas tout, certains éléments définissant cet objet, fonction ou relation étant dépendants de la configuration initiale du problème donné (dans le problème traite au chapitre 4, c'est le cas de l'objet *objet-cible*).

3.3.4. Règles

Les règles KIF permettent l'expression de règles d'inférence.

Règles monotoniques

Ce sont les règles d'inférence classiques de la logique élémentaire. Elles sont de la forme *(<<= <proposition> <proposition>*)* (ou sa forme inversée). Une règle KIF doit bien se distinguer d'une proposition composée au moyen d'un opérateur d'implication. En effet, prenons les deux paires d'expressions KIF suivantes :

(<<= (statut-connu ?x) (résident ?x))
(<<= (statut-connu ?x) (NOT (résident ?x)))

(<= (statut-connu ?x) (résident ?x))
(<= (statut-connu ?x) (NOT (résident ?x)))

Dans le premier cas (règles d'inférence), on ne peut inférer *(statut-connu Michel)* que si on a pu inférer soit *(résident Michel)* soit *(NOT résident Michel)*. Dans le deuxième cas, *(statut-connu Michel)* est forcément vrai puisqu'on peut le déduire d'une proposition et aussi de la proposition contraire.

Règles non-monotoniques

Ces règles permettent de faire des inférences "consistantes" lorsqu'on ne dispose pas de toute l'information nécessaire pour faire une inférence classique. On peut les exprimer grâce à l'opérateur *CONSIS*. Une proposition avec cet opérateur est de la forme suivante: *(CONSIS <proposition>)*. La signification de *CONSIS* est "Il est consistant de supposer que la proposition est vraie". Cela veut dire qu'on ne connaît pas la valeur de vérité de la proposition, mais que rien dans le reste du texte KIF ne permet de déduire qu'elle est fausse. La règle suivante permet de mieux comprendre l'utilisation de *CONSIS* :

(<<= (vole ?x) (oiseau ?x) (CONSIS (vole ?x)))

Cette règle dit que si l'objet dénoté par *?x* est un oiseau (c'est-à-dire qu'il vérifie la relation *oiseau*), et qu'il est consistant de supposer qu'il vole (c'est-à-dire que la proposition *(vole ?x)* ajoutée à l'ensemble des propositions déjà présentes dans le texte KIF ne rendrait pas celui-ci inconsistant), alors on peut inférer que l'objet dénoté par *?x* vole. En pratique, cette inférence signifie qu'on peut supposer qu'un oiseau de type donné vole, sauf s'il est explicitement mentionné par ailleurs qu'il ne vole pas.

3.3.5. Texte KIF

Un texte KIF est un texte composé :

- d'un ensemble de définitions qui permettent de définir les objets de l'univers du discours, les fonctions et les relations présents dans le système de connaissances
- d'un ensemble de propositions exprimant des faits sur le monde étudié
- d'un ensemble de règles exprimant les pas d'inférence légaux

L'un ou l'autre de ces ensembles peut être vide (certains problèmes ne demandent pas de décrire des règles d'inférence, par exemple). L'ordre des phrases du texte KIF est indifférent (KIF est un langage déclaratif).

3.4. EXPRESSION DE LA METACONNAISSANCE

REMARQUE PRELIMINAIRE : Cette section présente les concepts de représentation de la métaconnaissance en KIF. Cependant, notre solution de représentation de la métaconnaissance du problème du singe et des bananes, que nous proposerons au chapitre 6, n'utilise aucun de ces concepts. Le lecteur pressé peut donc sauter cette section, elle n'est pas indispensable pour la compréhension de la suite de l'exposé.

L'idée centrale de la représentation de la métaconnaissance en KIF est de concevoir les expressions du langage comme des *objets*. Une expression atomique est vue comme un objet primitif et une expression complexe comme une liste d'objets. Par exemple, l'expression *(NOT (= (+ a b) c))* peut être vue comme une liste de deux items, le premier étant l'opérateur *NOT* et le deuxième comme étant lui-même une liste de trois items, dans l'ordre : l'opérateur =, l'expression *(+ a b)* et l'atome *c* ; l'expression *(+ a b)* est elle-même vue comme une liste composée de la constante de fonction + et des objets *a* et *b*.

Il existe deux manières de dénoter les expressions comme des objets.

La première est d'utiliser l'opérateur *quote* devant une expression. Ainsi, si *(+ a b)* est une expression du langage ayant une valeur sémantique (puisqu'il s'agit d'un terme fonctionnel), le terme *'(+ a b)* dénote l'expression en tant que telle et permet de raisonner sur l'expression elle-même. Prenons par exemple la relation *(est-composé-de ?x ?y)* qui est vraie lorsque l'objet dénoté par *?x* est composé de la matière dénotée par *?y*. En soi, la proposition *(est-composé-de soleil fromage)* qui signifie que le soleil est composé de fromage, est évidemment fausse. Supposons maintenant que nous sommes intéressés par ce que les gens croient au sujet de la composition des objets, et que nous ayons la relation *(croit ?x ?y)* qui est vraie si l'individu dénoté par *?x* croit ce qui est dénoté par *?y*. Nous pouvons exprimer que l'individu *Eric* croit que le soleil est fait de fromage par la proposition suivante :

(croit Eric '(est-composé-de soleil fromage))

Si nous voulons raisonner sur les croyances des individus, nous pouvons exprimer que Marie ne croit rien de ce que croit Eric par la proposition suivante :

(=> (croit Eric ?p) (NOT (croit Marie ?p)))

dont nous pouvons déduire que Marie ne croit pas que le soleil soit composé de fromage.

La deuxième manière de dénoter les expressions comme des objets est d'utiliser l'opérateur *LISTOF* combiné au *quote*, afin de quantifier des parties d'expressions (qui sont vues comme des listes). L'expression *(+ a b)*, par exemple, qui peut être dénotée par le terme quoté *'(+ a b)* peut aussi être dénotée par le terme de liste *(LISTOF '+ 'a 'b)*. L'intérêt est que l'on peut décomposer une expression complexe en expressions plus simples et traiter différemment ces expressions en quantifiant certaines. Par exemple, supposons que Julie soit moins sceptique que Marie quant aux croyances d'Eric. Elle croit tout ce que croit Eric en ce qui concerne la composition des choses. On peut exprimer cela par la proposition suivante (dans lesquels les variables individuelles *?x* et *?y* sont, par défaut, quantifiées universellement) :

(=> (croit Eric (LISTOF 'est-composé-de ?x ?y))
(croit Julie (LISTOF 'est-composé-de ?x ?y)))

Cette proposition signifie que, quel que soient l'objet (*?x)* et le matériau (*?y*), si Eric croit que cet objet est composé de ce matériau, alors Julie le croit aussi.

Par ailleurs, pour pouvoir changer de niveau de dénotation, KIF offre quelques fonctions et relations. Citons les deux principales (pour plus de détails, voir [GENESERETH92], chapitre 9) :

- le terme fonctionnel (*denotation* τ) dénote l'objet dénoté par l'objet qui est lui-même dénoté par τ. Par exemple, *(denotation 'soleil)* dénote le soleil.
- la proposition relationnelle *(true p)* est vraie si l'expression dénotée par *p* est une proposition à valeur de vérité "vrai".

Pouvoir manipuler les expressions (qui sont de la connaissance) comme des objets, et exprimer des connaissances sur ces objets, c'est bien pouvoir exprimer la métaconnaissance d'un problème.

3.5. QUELQUES FONCTIONS ET RELATIONS

Dans cette dernière section, nous allons reprendre quelques fonctions et relations dont les définitions KIF sont données dans [GENESERETH92]. Ces fonctions et relations sont celles que nous utiliserons dans le prochain chapitre.

3.5.1. Fonctions

Rappelons qu'une fonction renvoie ⊥ si son résultat est indéterminé.

- ***(first ?l)*** : si *?l* dénote une liste non vide, le terme résultat de cette fonction dénote le premier item de *?l*
- ***(rest ?l)*** : si *?l* dénote une liste de plus d'un item, le terme résultat de cette fonction dénote la liste composée de *?l* de laquelle on a ôté le premier élément
- ***(last ?l)*** : si *?l* dénote une liste non vide, le terme résultat de cette fonction dénote le dernier item de *?l*
- ***(butlast ?l)*** : si *?l* dénote une liste de plus d'un item, le terme résultat de cette fonction dénote la liste composée de *?l* de laquelle on a ôté le dernier élément
- ***(union ?e1 ?e2)*** : si *?e1* et *?e2* dénotent des ensembles, le terme résultat de cette fonction dénote l'union de *?e1* et *?e2* (au sens mathématique du terme)
- ***(différence ?e1 ?e2)*** : si *?e1* et *?e2* dénotent des ensembles, le terme résultat de cette fonction dénote la différence de *?e1* et *?e2* (au sens mathématique du terme), c'est-à-dire *?e1* dont on a ôté l'intersection avec *?e2*
- ***(apply ?f ?l)*** : si *?f* dénote une fonction et *?l* une liste, le terme résultat de cette fonction dénote le résultat de l'application de *?f* aux items de la liste *?l*
- ***(append ?l1 ?l2)*** : si *?l1* et *?l2* dénotent des listes, le terme résultat de cette fonction dénote la liste composée, dans l'ordre, des items de *?l1* puis des items de *?l2*

3.5.2. Relations

- ***(member ?o ?e)*** : est vraie si *?o* dénote un élément appartenant à l'ensemble dénoté par *?e*
- ***(integer ?n)*** : est vraie si *?n* dénote un nombre entier
- ***(real-number ?n)*** : est vraie si *?n* dénote un nombre réel
- ***(>= ?n1 ?n2)*** : est vraie si *?n1* dénote un nombre supérieur ou égal au nombre dénoté par *?n2*
- ***(=< ?n1 ?n2)*** : est vraie si *?n1* dénote un nombre inférieur ou égal au nombre dénoté par *?n2*
- ***(subset ?e1 ?e2)*** : est vraie si *?e1* dénote un sous-ensemble de l'ensemble dénoté par *?e2*
- ***(list ?l)*** : est vraie si *?l* dénote une liste
- ***(item ?i ?l)*** : est vraie si *?i* dénote un item de la liste dénotée par *?l*
- ***(double ?l)*** : est vraie si *?l* dénote une liste de deux items
- ***(null ?l)*** : est vraie si *?l* dénote une liste vide (de 0 item)

CHAPITRE 4
KIF MIS EN PRATIQUE

4.1. INTRODUCTION

Pour évaluer l'intérêt des concepts du langage KIF dans la définition d'un langage de spécification abstraite d'un S.E., il est nécessaire de mettre ces concepts en pratique, ce qui veut dire confronter le langage à un problème concret de spécification d'un système de connaissances.

Nous allons donc effectuer cette mise en pratique des concepts de KIF en tentant de modéliser le problème du singe et des bananes, dont nous avons donné les bases informelles ainsi qu'une analyse des besoins au chapitre 2. Il faut se rappeler qu'au départ, KIF n'est pas conçu pour permettre une spécification formelle et abstraite à partir d'une modélisation informelle, mais bien pour transformer un système de connaissances écrit dans un langage donné (et formel) A en un texte abstrait KIF, pour permettre ensuite une traduction inverse dans un langage B, les deux textes en A et B étant sémantiquement identiques. La question sous-jacente à l'ensemble de ce chapitre n'est donc pas "KIF est-il un bon langage de spécification abstraite des systèmes de connaissances ?" mais plutôt "Quels sont les concepts nécessaires à la spécification abstraite des systèmes de connaissances que je peux représenter en KIF ?". Nous tenterons de répondre à cette question dans le chapitre 4, à la lumière de la mise en pratique qui suit.

4.2. DEFINITION DES OBJETS DU PROBLEME

REMARQUE PRELIMINAIRE : il y a une difficulté d'expression des concepts du problème, parce que le terme d' "objet" désigne à la fois le concept d'objet en KIF et celui d'objet physique inanimé (le lit, la couverture, les bananes, l'échelle). Ces derniers seront donc constamment appelés "objets physiques" ou "objets physiques inanimés" tandis que les premiers seront appelés "objets KIF". Utilisé seul, le terme "objet" désignera les objets du problème, c'est-à-dire les objets physiques et le singe.

Il existe deux types d'objets dans le problème du singe et des bananes. Le premier type regroupe les bananes, la couverture, le lit et l'échelle. Ce sont les objets physiques inanimés, incapables d'effectuer des actions par eux-mêmes dans le système. Comme ils sont plusieurs, on peut considérer qu'il y a une classe d'objets physiques et que chaque objet physique particulier est une instance de cette classe[14]. Ces notions de classe et d'instance sont celles des modèles orientés-objet. Le deuxième type d'objet est l'objet mobile autonome, en l'occurrence le singe,

[14]En pratique, comme on le verra, la classe d'objets physiques sera représentée par un ensemble d'objets physiques, les instances de cette classe étant les éléments de l'ensemble. Nous ferons donc un abus de langage en appelant cette classe indifféremment "classe des objets physiques" ou "ensemble des objets physiques".

qui est capable d'effectuer un certain nombre d'actions. Cet objet étant unique, nous ne le représenterons pas comme instance de classe (une telle modélisation serait envisageable, mais alourdirait les écritures en KIF).

4.2.1. Définition des objets physiques

Un objet physique est entièrement défini par quatre éléments[15] :

- un *nom*, qui est unique (puisqu'il n'existe qu'un seul objet de nom donné dans le système) et qui identifie donc l'objet ;
- une *position horizontale*, donnée par un couple de valeurs entières comprises entre 1 et 10 qui représentent les coordonnées de l'objet par rapport au sol ;
- une *position verticale*, qui appartient à l'ensemble de valeurs suivant : {sol, plafond, porté, sur(autre-objet)}. "porté" signifie que l'objet est porté par le singe et "sur(autre-objet)" signifie qu'il est posé sur l'autre objet indiqué ;
- un *poids* qui appartient à l'ensemble {lourd, léger}. "lourd" signifie "trop lourd pour pouvoir être porté par le singe".

Le produit cartésien des objets physiques

Afin de circonscrire entièrement les différents états possibles des objets physiques du problème, nous définirons l'objet KIF "produit cartésien des objets physiques" qui est l'ensemble de toutes les combinaisons possibles des éléments qui composent ces objets physiques. Il s'agit en quelque sorte de l' "univers" des objets physiques, de l'ensemble des valeurs qui sont *a priori* susceptibles de représenter un objet physique particulier. Voici la définition KIF de ce produit cartésien :

```
(DEFOBJECT produit-cartésien-des-objets-physiques :=
    (SETOFALL (LISTOF ?nom ?pos-hor ?pos-ver ?poids)
              (AND (member ?nom (SETOF 'bananes 'lit 'couverture 'échelle))
                   (= ?pos-hor (LISTOF ?abs ?ord))
                   (integer ?abs) (>= ?abs 1) (=< ?abs 10)
                   (integer ?ord) (>= ?ord 1) (=< ?ord 10)
                   (member ?pos-ver (SETOF 'sol 'plafond 'porté @nom2))
                   (= (SETOF @nom2) (SETOF 'bananes 'lit 'couverture 'échelle))
                   (member ?poids 'lourd 'léger))))
```

En français[16] : le produit cartésien des objets physiques est l'ensemble de toutes les listes de quatre items[17] dont le premier est un nom appartenant à l'ensemble {bananes, lit, couverture, échelle} ; le deuxième est une position horizontale de la forme [abscisse, ordonnée] où abscisse et ordonnée sont des coordonnées entières comprises entre 1 et 10 ; le troisième est une position verticale appartenant à l'ensemble {sol, plafond, porté, nom d'objet}, "porté" signifiant que l'objet est entre les mains du singe et "nom d'objet" étant le nom de l'objet physique sur lequel l'objet physique représenté (dont le nom est donné par le

---

[15]Pour plus de détails sur cette définition, voir 2.2.3.

[16]Toute explication précédée de "En français" signifie que nous allons expliquer la phrase KIF qui précède (et non donner une vision informelle de l'objet défini par ce texte).

[17]Bien que le terme "item" soit moins élégant que le terme "élément", nous l'utiliserons pour désigner un élément de liste, par opposition aux éléments qui appartiennent a un ensemble.

premier item de la liste) est posé ; le dernier item est un poids appartenant à l'ensemble {lourd, léger}.

Cette définition est complète (voir chapitre 3) : elle circonscrit entièrement l'univers des objets physiques. Toutes les représentations possibles d'un objet physique sont contenues dans l'ensemble *produit-cartésien-des-objets-physiques*, même celles qui n'ont pas de sens, comme [bananes, [5,5], bananes, léger] qui indiquerait que les bananes sont posées sur elles-mêmes. Il n'y a en effet aucune contrainte sur les items des listes de cet ensemble, si ce n'est la définition des ensembles auxquels ils doivent appartenir. Les contraintes d'intégrité interviendront dans la définition des composantes réelles du problème (ici, la classe des objets physiques).

La classe des objets physiques

La représentation de la classe des objets physiques n'est pas possible en tant qu'objet KIF du problème, par une définition *DEFOBJECT*. Contrairement au produit cartésien que nous venons de définir, cette classe d'objets évolue au cours de la résolution du problème. En effet, les objets physiques qui la composent bougent suivant ce que le singe fait d'eux. Comme la position spatiale de ces objets est une composante de leur définition, cette définition change au cours du temps.

La solution que nous avons adoptée est de représenter l'ensemble des objets physiques sous la forme d'une relation au sens KIF du terme, relation exprimée par *est-l-ensemble-des-objets-physiques (?ens-obj)*. Cette relation est en fait l'expression KIF d'un prédicat unaire et signifie que l'objet KIF désigné par *?ens-obj* est l'ensemble des objets physiques du problème. Pour être précis, c'est l'objet KIF vérifiant cette relation qui représente l'ensemble des objets physiques.

La classe des objets physiques est un sous-ensemble du produit cartésien de ces objets respectant un ensemble de contraintes qu'on peut appeler *contraintes d'intégrité* ou *invariants* de la classe des objets physiques. Avant de définir la relation *est-l-ensemble-des-objets-physiques*, nous allons répertorier ces invariants et les définir sous la forme de relations KIF :

- **contrainte de complétude** : c'est la contrainte qui impose à la classe des objets physiques de contenir une représentation de chacun de ces objets physiques.
- **contrainte d'unicité** : c'est la contrainte qui impose à la classe des objets physiques de ne contenir qu'une seule représentation de chaque objet physique. Les contraintes de complétude et d'unicité garantissent un isomorphisme entre les objets physiques et leur représentation.
- **autres invariants** : nous n'allons pas tenter d'en dresser une liste exhaustive[18]. Citons-en tout de même quelques-uns :
  - (I1) Aucun objet physique ne peut être posé sur un objet physique qui est lui-même au plafond ou porté par le singe.
  - (I2) Il ne peut exister de listes circulaires d'objets physiques posés l'un sur l'autre (a est posé sur b qui est posé sur c qui est posé sur a).

[18]Il est évident que l'ingénieur chargé de la spécification réelle de ce problème devra, quant à lui, déterminer de façon exhaustive ces invariants et les décrire en KIF. Notre propos n'étant pas là, il nous suffit de montrer que cette description est possible, et comment la réaliser.

(I3) Un objet physique posé sur un autre est à la même position horizontale que celui-ci.

Les définitions KIF associées aux contraintes de complétude (*invariant-ens-obj-1*), d'unicité (*invariant-ens-obj-2*) et à l'invariant (I3) de la liste ci-dessus (*invariant-ens-obj-3*) sont présentées à titre d'exemple[19] :

```
(DEFRELATION invariant-ens-obj-1 (?ens-obj) :=
    (FORALL (?nom1)
            (=> (member ?nom (SETOF 'bananes 'lit 'couverture 'échelle))
                (EXISTS (?obj) (AND (member ?obj ?ens-obj)
                                    (= (nom ?obj) ?nom1))))))

(DEFRELATION invariant-ens-obj-2 (?ens-obj) :=
    (FORALL (?obj1 ?obj2)
            (=> (member ?obj1 ?ens-obj)
                (member ?obj2 ?ens-obj)
                (/= ?obj1 ?obj2)
                (/= (nom ?obj1) (nom ?obj2))))))

(DEFRELATION invariant-ens-obj-3 (?ens-obj) :=
    (FORALL (?obj1)
            (=> (member ?obj1 ?ens-obj)
                (OR (= (pos-ver ?obj1) 'plafond)
                    (= (pos-ver ?obj1) 'porté))
                (FORALL (?obj2)
                        (=> (member ?obj2 ?ens-obj)
                            (/= ?obj2 ?obj1)
                            (/= (pos-ver ?obj2) (nom ?obj1)))))))
```

REMARQUE : Prise isolément, la valeur de vérité d'un invariant n'a pas de sens car *a priori* l'argument de la relation peut valoir n'importe quoi. Par exemple, si *?ens-obj* est un objet KIF quelconque sauf un ensemble, on peut vérifier que la relation *invariant-ens-obj-2* est toujours vraie. Définir ces invariants n'a de sens que si on les intègre à la définition de la relation qui se rapporte à l'objet dont ils sont les invariants (ci-dessous).

Disposant de la définition du produit cartésien des objets physiques et des relations exprimant les invariants liés a ces objets physiques, nous pouvons définir la classe (ou l'ensemble) des objets physiques sous forme d'une relation KIF :

```
(DEFRELATION est-l-ensemble-des-objets-physiques (?ens-obj) :=
    (AND (subset ?ens-obj produit-cartésien-des-objets)
         (invariant-ens-obj-1 ?ens-obj) ... (invariant-ens-obj-n ?ens-obj)))
```

En français : un objet KIF quelconque représente l'ensemble des objets physiques si et seulement si :

- il est un sous-ensemble du produit cartésien de ces objets
- il vérifie tous les invariants décrits pour l'ensemble des objets physiques

L'instance "objet physique"

Nous pouvons enfin définir une relation est-un-objet-physique de la façon suivante :

---

[19]Certaines fonctions utilisées ici ((*nom ?obj*), (*pos-ver ?obj*), etc.) seront définies à la section suivante.

```
(DEFRELATION est-un-objet-physique (?obj) :=
    (EXISTS (?ens-obj) (AND (est-l-ensemble-des-objets-physiques ?ens-obj)
                            (member ?obj ?ens-obj))))
```

En français : un objet KIF représente un objet physique si et seulement s'il est élément d'un objet KIF qui représente quant à lui l'ensemble des objets physiques[20].

4.2.2. Définition de l'objet mobile (le singe)

Le singe est entièrement défini par :

- une *position horizontale*, donnée par un couple de valeurs entières comprises entre 1 et 10 qui représentent les coordonnées du singe par rapport au sol (similaire a la position horizontale d'un objet) ;
- une *position verticale*, qui appartient a l'ensemble {sol, bananes, lit, couverture, échelle}. "Sol" signifie que le singe est sur le sol. Chacune des autres valeurs signifie que le singe est sur l'objet physique représenté par cette valeur ;
- une information sur l'*objet porté* par le singe. Cette information appartient à l'ensemble {rien, bananes, lit, couverture, échelle}. "Rien" signifie que le singe n'a rien entre les mains. Chacune des autres valeurs signifie que le singe porte l'objet physique représenté par cette valeur.

Le produit cartésien du singe

De même que nous avons défini un produit cartésien pour les objets physiques, nous définissons un *produit-cartésien-du-singe* représentant l'ensemble des configurations possibles de ce singe. En voici la définition en KIF :

```
(DEFOBJECT produit-cartésien-du-singe :=
    (SETOFALL (LISTOF ?pos-hor ?pos-ver ?objet-porté)
              (AND (= ?pos-hor (LISTOF ?abs ?ord))
                   (integer ?abs) (>= ?abs 1) (=< ?abs 10)
                   (integer ?ord) (>= ?ord 1) (=< ?ord 10)
                   (member ?pos-ver (SETOF 'sol @nom-d-objet))
                   (member ?objet-porté (SETOF 'rien @nom-d-objet))
                   (= (SETOF @nom-d-objet)
                      (SETOF 'bananes 'lit 'couverture 'échelle)))))
```

En français : le produit cartésien du singe est l'ensemble de toutes les listes de trois items dont le premier est une position horizontale de la forme [abscisse, ordonnée] où abscisse et ordonnée sont des coordonnées entières comprises entre 1 et 10 ; le deuxième est une position verticale appartenant à l'ensemble {sol, nom d'objet}, "sol" signifiant que le singe est sur le sol, "nom d'objet" signifiant qu'il se trouve sur l'objet physique ainsi nommé ; le troisième item est une information sur ce que le singe porte et appartient à l'ensemble {rien, nom d'objet}, "rien" signifiant que le singe ne porte rien, "nom d'objet" signifiant que le singe porte l'objet physique ainsi nommé. Dans ces deux derniers items, "nom d'objet" appartient à l'ensemble {bananes, lit, couverture, échelle}.

---

[20]Nous aurions pu intégrer à cette définition certains invariants propres à un objet physique déterminé (par exemple: le lit ne peut pas être au plafond) mais nous préférons les intégrer à la définition de la relation *est-l'ensemble-des-objets-physiques*. Ce choix de modélisation permet d'avoir un ensemble des objets physiques déjà parfaitement valide par rapport aux spécifications du problème.

Le *produit-cartésien-du-singe* circonscrit entièrement l'ensemble des configurations possibles du singe. C'est la description de la relation *est-le-singe* (ci-dessous) qui permettra d'identifier un objet KIF comme représentant le singe, en respectant les invariants de celui-ci.

Le singe

Nous pouvons maintenant définir la relation *est-le-singe* comme suit :

```
(DEFRELATION est-le-singe (?singe) :=
    (AND (member ?singe produit-cartésien-du-singe)
         (invariant-singe-1 ?singe) ... (invariant-singe-n ?singe)))
```

En français : un objet KIF est une représentation du singe si et seulement si :

- il est un élément du produit cartésien du singe
- il vérifie les invariants décrits pour le singe[21]

4.2.3. Définition des états du problème

Il nous reste à définir une relation particulière que nous appellerons *est-un-état-du-problème* et qui représentera l'état courant des différents objets intervenant dans le problème (le singe et les objets physiques) :

```
(DEFRELATION est-un-état-du-problème (?état) :=
    (EXISTS (?singe ?ens-obj)
            (AND (= ?état LISTOF ?singe ?ens-obj)
                 (est-le-singe ?singe)
                 (est-l-ensemble-des-objets-physiques ?ens-obj)
                 (invariant-état-1 ?état) ... (invariant-état-n ?état))))
```

En français : un objet KIF est une représentation de l'état du problème si et seulement si :

- il a la forme d'une liste de deux items
- le premier de ces items est un objet KIF qui représente le singe
- le deuxième de ces items est un objet KIF qui représente l'ensemble des objets physiques
- il vérifie les invariants décrits sur l'état du problème

Les invariants sur l'état du problème sont tous ceux qui font intervenir à la fois le singe et les objets physiques. En voici quelques exemples :

- le singe ne peut porter que des objets de poids "léger"
- un objet porté par le singe est à la même position horizontale que celui-ci
- si le singe est sur un objet physique, il est à la même position horizontale que cet objet

---

[21] La spécification informelle du problème ne renseigne aucun invariant particulier qui se rapporte au singe. Tout élément du produit cartésien du singe décrit celui-ci de façon plausible La liste d'invariants sur le singe est donc *a priori* vide. Si la spécification informelle avait contenu par exemple la contrainte "Si le singe porte un objet, il ne peut se trouver que sur le sol", alors on aurait eu un invariant propre au singe. D'autre part, la spécification formelle peut être plus complète que la spécification informelle. Il est donc intéressant de conserver cette notion d'invariant sur le singe.

Cette définition nous permet d'affirmer que si un objet KIF respecte la relation est-un-état-du-problème, alors il représente un état cohérent par rapport aux spécifications du problème.

4.3. QUELQUES FONCTIONS UTILES

Nous allons définir ici quelques fonctions qui nous permettront d'alléger les écritures des définitions KIF, en particulier celles des fonctions de changement d'état (section 4.6.).

La fonction *nom* renvoie le nom d'un objet physique :

```
(DEFFUNCTION nom (?objet) :=
    (IF (member ?objet produit-cartésien-des-objets-physiques)
        (first ?objet)))
```

La fonction *pos-hor* renvoie la position horizontale d'un objet physique ou du singe :

```
(DEFFUNCTION pos-hor (?objet) :=
    (COND ( (member ?objet produit-cartésien-des-objets-physiques) (first (rest ?objet)))
          ( (member ?objet produit-cartésien-du-singe) (first ?objet))))
```

La fonction *pos-ver* renvoie la position verticale d'un objet physique ou du singe :

```
(DEFFUNCTION pos-ver (?objet) :=
    (COND ((member ?objet produit-cartésien-des-objets-physiques)
           (last (butlast ?objet)))
          ((member ?objet produit-cartésien-du-singe) (first (rest ?objet)))))
```

La fonction *poids* renvoie le poids d'un objet physique :

```
(DEFFUNCTION poids (?objet) :=
    (IF (member ?objet produit-cartésien-des-objets-physiques)
        (last ?objet)))
```

La fonction *objet-porté* renvoie le nom de l'objet porté par le singe :

```
(DEFFUNCTION objet-porté (?singe) :=
    (IF (member ?singe produit-cartésien-des-objets-physiques) (last ?singe)))
```

La fonction *objet-de-nom* renvoie l'objet physique dont le nom est donné en argument :

```
(DEFFUNCTION objet-de-nom (?nom-d-objet) :=
    (IF (EXISTS (?objet) (AND (est-un-objet-physique ?objet)
                              (= ?nom-d-objet (nom ?objet))))
        ?objet))
```

Remarque au sujet de cette fonction : l'unicité du résultat n'est pas garantie directement par la définition de la fonction elle-même, mais par la définition de la relation *est-un-objet-physique* qui garantit (contrainte d'unicité) qu'il n'y a qu'un exemplaire d'objet physique de nom donné dans un ensemble qui vérifie cette relation. Dès lors, s'il existe un objet appartenant à l'ensemble des objets physiques dont le nom est égal à un nom donné, cet objet est nécessairement unique. Mais puisque pour garantir l'unicité du résultat, il faut utiliser une relation (*est-un-objet-*

physique) qui implique que les invariants sur l'ensemble des objets physiques soient préalablement définis, il est clair qu'aucun de ces invariants ne pourra utiliser la fonction *objet-de-nom* (pour éviter les définitions croisées).

La fonction *état-du-singe* renvoie le singe tel qu'il est représenté dans l'état du problème :

```
(DEFFUNCTION état-du-singe (?état) :=
    (IF (est-un-état-du-problème ?état)
        (first ?état)))
```

La fonction *état-des-objets-physiques* renvoie l'ensemble des objets physiques tel qu'il est représenté dans l'état du problème :

```
(DEFFUNCTION état-des-objets-physiques (?état) :=
    (IF (est-un-état-du-problème ?état)
        (last ?état)))
```

4.4. DEFINITION DE LA CONFIGURATION INITIALE DU PROBLEME

Pour définir la configuration initiale du problème, nous avons deux façons de procéder : la première consiste à définir complètement l'état initial du problème, la seconde à le définir partiellement.

4.4.1. Variante I : Définition complète

Définir la configuration initiale (ou état initial) du problème correspond à définir un objet "état initial" qui respecte la relation *est-un-état-du-problème*. Voici un exemple de texte KIF correspondant à cette définition :

```
(DEFOBJECT état-initial-du-problème :=
    (LISTOF (LISTOF (LISTOF 5 10) 'sol 'couverture)
            (SETOF (LISTOF 'bananes (LISTOF 2 3) 'plafond 'léger)
                   (LISTOF 'lit (LISTOF 1 1) 'sol 'lourd)
                   (LISTOF 'couverture (LISTOF 5 10) 'porté 'léger)
                   (LISTOF 'échelle (LISTOF 1 1) 'lit 'léger))))
```

Cette définition KIF de l'état initial du problème signifie que, dans la configuration initiale :

- le singe[22] se trouve en (5,10), au sol et il tient la couverture
- les bananes sont en (2,3), au plafond, et sont de poids léger
- le lit est en (1,1), au sol, et est de poids lourd
- la couverture est en (5,10), portée par le singe, et est de poids léger
- l'échelle est en (1,1), sur le lit et est de poids léger

Le texte KIF contient également la proposition

```
(est-un-état-du-problème état-initial-du-problème)
```

[22]La représentation du singe dans l'état initial du problème ne contient pas l'information nominative "singe". Il s'agit évidemment du premier item décrit dans cette définition.

qui signifie que l'état initial respecte bien la structure et les invariants de l'état du problème et des objets qui le composent.

Avantages de cette variante : l'état initial est complètement défini.

Inconvénients : chaque redéfinition de l'état initial demande une nouvelle définition de l'objet *état-initial-du-problème* ; la validité de l'état initial n'est pas garantie par sa définition.

4.4.2. Variante II: Définitions partielles

Dans cette variante, on donne d'abord une définition partielle de l'état initial du problème. Cette définition mentionne simplement qu'il s'agit d'un état du problème :

```
(DEFOBJECT état-initial-du-problème
    (est-un-état-du-problème état-initial-du-problème))
```

L'état initial est complètement cerné par une deuxième définition partielle qui donne les valeurs des objets qui composent cet état :

```
(DEFOBJECT état-initial-du-problème
    (= (état-du-singe état-initial-du-problème)
       (LISTOF (LISTOF 5 10) 'sol 'couverture))
    (member (LISTOF 'bananes (LISTOF 2 3) 'plafond 'léger)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'lit (LISTOF 1 1) 'sol 'lourd)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'couverture (LISTOF 5 10) 'porté 'léger)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'échelle (LISTOF 1 1) 'lit 'léger)
            (état-des-objets-physiques état-initial-du-problème)))
```

La première définition partielle exprime que l'état initial du problème est un état particulier de celui-ci, donc qu'il a bien la structure d'une liste de deux items, le premier - *(état-du-singe état-initial-du-problème)* dans la deuxième définition - représentant le singe, le second - *(état-des-objets-physiques état-initial-du-problème)* dans la deuxième définition - représentant l'ensemble des objets physiques inanimés. Cette définition est vraie quel que soit l'état initial réel du problème et est toujours présente telle quelle dans le texte KIF.

La deuxième définition "attribue" des valeurs aux éléments qui composent l'état initial du problème. Elle varie donc suivant la configuration initiale qu'on veut lui donner. Dans l'exemple ci-dessus, cette configuration est identique à celle proposée dans la variante I.

Avantages de cette variante : le problème contient une définition de l'état initial indépendante de la valeur de cet état, ce qui donne à la spécification une plus grande réutilisabilité et une plus grande modularité ; la validité de l'état par rapport aux invariants du problème est contenue dans cette définition.

Inconvénients : rien ne garantit qu'il n'y a pas contradiction entre les deux définitions partielles de l'état initial, ni que ces deux définitions cernent complètement un état donné.

Les deux variantes se valent. Pour une question d'uniformité avec la définition du but du problème (ci-dessous), nous adopterons la seconde.

4.5. DEFINITION DU BUT DU PROBLEME

Le but du problème est, comme on l'a vu, que le singe se saisisse d'un objet que l'on aura désigné comme objet-cible (dans l'exemple de départ, la banane). Quel que soit cet objet-cible, il répond au moins à deux conditions : premièrement, il doit être un objet physique présent dans le problème ; deuxièmement, son poids doit être "léger" puisque le singe doit pouvoir le tenir dans ses mains. On peut donc lui donner la définition partielle suivante :

```
(DEFOBJECT objet-cible
    (est-un-objet-physique objet-cible)
    (= (poids objet-cible) 'léger))
```

Cette définition partielle de l'objet-cible a l'avantage d'être vraie quel que soit l'objet-cible réellement désigné.

Pour désigner l'objet-cible réel du problème, on donne une seconde définition partielle qui, elle, dépend véritablement du problème posé :

```
(DEFOBJECT objet-cible
    (= (nom objet-cible) 'bananes))
```

En associant ces deux définitions partielles, on cerne complètement l'objet-cible du problème : en effet, la relation *(est-un-objet-physique objet-cible)* garantit qu'il n'y a pas d'autre objet de même nom que l'objet-cible dans le système. En donnant explicitement ce nom dans la deuxième définition partielle, on désigne donc l'objet-cible de façon unique. Les deux définitions doivent bien sûr ne pas être contradictoires (par exemple, l'objet désigné ne peut pas être un objet lourd).

Le but du problème est atteint lorsque le système a généré un état du problème dans lequel le singe tient l'objet désigné comme objet cible. Cet état n'est évidemment pas unique. Nous pouvons toutefois définir l'*ensemble des états-buts* de la manière suivante :

```
(DEFOBJECT ensemble-des-états-buts :=
    (SETOFALL (?état-but)
              (AND (est-un-état-du-problème ?état-but)
                   (= (objet-porté (état-du-singe ?état-but)) (nom objet-cible)))))
```

En français : l'ensemble des états-buts est l'ensemble des objets KIF qui :

- représentent un état du problème
- contiennent, dans cette représentation, l'information que le singe porte l'objet-cible

Sans trop anticiper sur la suite, il est clair que la solution générée par le système devra produire un état qui appartient à cet ensemble des états-buts. Nous pouvons concrétiser cette notion d' "état-solution" par la relation KIF suivante :

```
(DEFRELATION est-état-solution-du-problème (?état) :=
    (member ?état ensemble-des-états-buts))
```

4.6. DEFINITION DES FONCTIONS DE CHANGEMENT D'ETAT

Les fonctions de changement état correspondent aux actions que le singe peut entreprendre[23]. A partir d'un état du problème, l'application d'une de ces fonctions - pour peu qu'elle puisse être appliquée - produit un nouvel état. Si ces fonctions sont correctement spécifiées et que l'état de départ est valide (c'est-à-dire qu'il vérifie la relation *est-un-état-du-problème*), le nouvel état généré par la fonction sera valide lui aussi.

Ces fonctions sont donc des actions dont l'exécution est conditionnée par un certain nombre de préconditions. Si ces préconditions sont vérifiées, la fonction génère un état résultant de l'exécution de l'action qu'elle décrit.

Les fonctions de changement d'état ont divers arguments (objet physique, position du mouvement) et en ont toutes un en commun, qui est l'état du problème sur lequel elles sont appliquées. Cet état doit bien sûr être valide. Dès lors, outre les préconditions spécifiques à chaque fonction, il y aura une précondition commune à toutes : l'état de départ doit être valide. On peut exprimer cette précondition par la relation *est-un-état-du-problème (?état)* ou *?état* est l'argument de la fonction qui représente l'état de départ.

Dans la description de chacune des fonctions de changement d'état, nous donnerons la spécification informelle de la fonction puis son texte en KIF. La spécification informelle sous forme de pré- et postconditions est celle du texte informel de la spécification [BUSSENOT92].

4.6.1. Fonction *saisir-objet (?nom-obj)*

C'est la fonction par laquelle le singe prend un objet entre les mains.

Préconditions[24] :

- le singe ne porte rien
- *?nom-obj* est le nom d'un objet physique
- *?nom-obj* a un poids léger
- *?nom-obj* et le singe ont la même position horizontale
- il n'y a rien sur *?nom-obj*
- le singe est sur le sol et *?nom-obj* n'est pas au plafond **ou bien** le singe est sur l'échelle et *?nom-obj* est au plafond

Postconditions :

- le singe porte *?nom-obj*
- la position verticale de *?nom-obj* est "porté"

---

[23]Nous utiliserons d'ailleurs indifférement les termes "action" et "fonction de changement d'état" dans la suite de cet exposé.

[24]La réponse à la question "Que produit une fonction dont les préconditions ne sont pas respectées ?" est traitée après la description de toutes les fonctions.

Définition KIF :

```
(DEFFUNCTION saisir-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (poids (objet-de-nom ?nom-obj)) 'léger)
             (= (pos-hor (état-du-singe ?état)) (pos-hor (objet-de-nom ?nom-obj)))
             (FORALL (?obj2) (=> (est-un-objet-physique ?obj2)
                                 (/= (pos-ver ?obj2) ?nom-obj)))
             (OR (AND (= (pos-ver (état-du-singe ?état)) 'sol)
                      (/= (pos-ver (objet-de-nom ?nom-obj)) 'plafond))
                 (AND (= (pos-ver (état-du-singe ?état)) 'échelle)
                      (= (pos-ver (objet-de-nom ?nom-obj)) 'plafond))))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        (pos-ver (état-du-singe ?état))
                        ?nom-obj)
                (union (difference (état-des-objets-physiques ?état)
                                   (SETOF (objet-de-nom ?nom-obj)))
                       (SETOF (LISTOF ?nom-obj
                                      (pos-hor (objet-de-nom ?nom-obj))
                                      porté
                                      (poids (objet-de-nom ?nom-obj))))))))
```

En français : appliquée aux arguments *?état* et *?nom-obj*, et si

- *?état* représente un état du problème ET
- *?nom-obj* représente un objet physique (*objet-de-nom (?nom-obj)*) ET
- *?nom-obj* est de poids léger ET
- *?nom-obj* et la représentation du singe dans *?état* sont à la même position horizontale ET
- pour tout objet physique, sa position verticale n'est pas sur *?nom-obj* ET
- OU BIEN la représentation du singe dans *?état* est au sol et *?nom-obj* n'est pas au plafond OU BIEN la représentation du singe dans *?état* est sur l'échelle et *?nom-obj* est au plafond,

alors la fonction *saisir-objet* renvoie une liste de deux items, le premier représentant le singe avec la même position horizontale et la même position verticale que dans *?état*, et avec *?nom-obj* pour objet porté, le deuxième représentant l'ensemble des objets physiques dont on ôte ("difference") *?nom-obj* et auquel on rajoute ("union") une liste représentant le nouvel état de l'objet identifié par *?nom-obj*, liste contenant son nom, sa position horizontale, "porté" pour position verticale, et son poids.

On peut constater que si les préconditions sont vérifiées, cette fonction produit une liste de deux items qui respecte pleinement la structure d'un état du problème. Si cette fonction est valide (et elle l'est), cet état résultant est un nouvel état du problème, c'est-à-dire qu'il vérifie bien la relation *est-un-état-du-problème*.

4.6.2. Fonction *lâcher-objet (?nom-obj)*

C'est la fonction qui fait lâcher au singe l'objet qu'il tient en main. On considérera que, même s'il y a un autre objet physique à l'endroit où le singe lâche celui qu'il a en main, l'objet lâché tombera sur le sol (ce qui suppose que parmi les invariants de l'ensemble des objets physiques, il n'y en ait pas qui dise "deux objets ne peuvent être à la fois à la même position horizontale et verticale". Si c'était le cas, la fonction ci-dessous serait incorrecte, puisqu'elle pourrait produire un état non valide).

Préconditions :

- *?nom-obj* est le nom d'un objet physique (*objet-de-nom (?nom-obj)*)
- le singe tient *?nom-obj*

Postconditions :

- le singe ne porte plus rien
- la position verticale de *?nom-obj* est le sol

Définition KIF :

```
(DEFFUNCTION lâcher-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (objet-porté (état-du-singe ?état)) ?nom-obj))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        (pos-ver (état-du-singe ?état))
                        'rien)
                (union (difference (état-des-objets-physiques ?état)
                                   (SETOF (objet-de-nom ?nom-obj)))
                       (SETOF (LISTOF ?nom-obj
                                      (pos-hor (objet-de-nom ?nom-obj))
                                      'sol
                                      (poids (objet-de-nom ?nom-obj))))))))
```

En français : appliquée aux arguments *?état* et *?nom-obj*, et si

- *?état* représente un état du problème ET
- *?nom-obj* représente un objet physique ET
- l'objet porté par la représentation du singe dans *?état* est bien *?nom-obj*,

alors la fonction *lâcher-objet* renvoie une liste de deux items, le premier représentant le singe avec la même position horizontale et la même position verticale que dans *?état*, et avec "rien" comme information sur ce qu'il porte, le deuxième représentant l'ensemble des objets physiques dont on ôte ("difference") l'objet identifié par *?nom-obj* et auquel on rajoute ("union") une liste représentant le nouvel état de cet objet, liste contenant son nom, sa position horizontale, "sol" pour position verticale, et son poids.

4.6.3. Fonction *aller-en (?x ?y)*

C'est la fonction par laquelle le singe se déplace au point de coordonnées (x,y).

Préconditions[25] :

- *?x, ?y* sont des coordonnées plausibles (entre 1 et 10)
- le singe est sur le sol

Postconditions :

- SI le singe ne porte rien ALORS
  - le singe est en *(?x, ?y)*
- SI le singe porte un objet ALORS
  - le singe est en *(?x, ?y)*
  - l'objet est en *(?x, ?y)*

Définition KIF :

```
(DEFFUNCTION aller-en (?état ?x ?y) :=
    (IF (AND (est-un-état-du-problème ?état)
             (integer ?x) (>= ?x 1) (=< ?x 10)
             (integer ?y) (>= ?y 1) (=< ?y 10)
             (= pos-ver (état-du-singe (?état)) 'sol))
        (IF (= (objet-porté (état-du-singe ?état)) 'rien)
            (LISTOF (LISTOF (LISTOF ?x ?y)
                            (pos-ver (état-du-singe ?état))
                            (objet-porté (état-du-singe ?état)))
                    (état-des-objets-physiques ?état))
            (LISTOF (LISTOF (LISTOF ?x ?y)
                            (pos-ver (état-du-singe ?état))
                            (objet-porté (état-du-singe ?état)))
                    (union (difference (état-des-objets-physiques ?état)
                                       (SETOF
                                        (objet-de-nom (objet-porté (état-du-singe ?état)))))
                           (SETOF (LISTOF (objet-porté (état-du-singe ?état))
                                          (LISTOF ?x ?y)
                                          'porté
                                          (poids (objet-de-nom
                                                  (objet-porté
                                                   (état-du-singe ?état))))))))))
```

En français : appliquée aux arguments *?état*, *?x* et *?y*, et si

- *?état* représente un état du problème ET
- *?x* et *?y* sont des valeurs entières comprises entre 1 et 10 ET
- la position verticale de la représentation du singe dans *?état* est "sol" ET
- la position horizontale du singe n'est pas [*?x*, *?y*]

alors la fonction *aller-en* renvoie :

- si l'objet porté par la représentation du singe dans *?état* est "rien" (il ne porte pas d'objet), une liste de deux items, le premier représentant le singe avec comme position horizontale [*?x*, *?y*], le deuxième représentant l'ensemble des objets physiques inchangé

[25]Nous n'imposerons pas comme précondition que le singe soit ailleurs qu'en (x,y). Si c'est le cas, la fonction est applicable mais n'a aucun effet sur état.

- sinon (le singe porte un objet), une liste de deux items, le premier représentant le singe avec comme position horizontale [*?x*, *?y*], le deuxième représentant l'ensemble des objets physiques dont on a ôté l'élément représentant l'objet porté par le singe et auquel on a rajouté un élément représentant ce même objet modifié, avec [*?x*, *?y*] pour position horizontale

4.6.4. Fonction *mettre-objet-en (?nom-obj ?x ?y)*

C'est la fonction par laquelle le singe place un objet donné à une position (x,y). Elle correspond à la composition de la fonction *aller-en ?x ?y* dans le cas particulier où le singe tient un objet, et de la fonction *lâcher-objet ?nom-obj*.

Préconditions :

- préconditions de *aller-en (?x ?y)*
- *?nom-obj* est un nom d'objet physique
- le singe tient *?nom-obj*

Postconditions :

- la position horizontale du singe est *(?x, ?y)*
- la position horizontale de *?nom-obj* est *(?x, ?y)*
- le singe ne porte plus rien
- la position verticale de *?nom-obj* est le sol

Définition KIF :

```
(DEFFUNCTION mettre-objet-en (?état ?nom-obj ?x ?y) :=
    (lâcher-objet (aller-en ?état ?x ?y) ?nom-obj))
```

En français : la fonction *(mettre-objet-en ?état ?nom-obj ?x ?y)* correspond à appliquer la fonction *lâcher-objet* à l'état produit par la fonction *(aller-en ?état ?x ?y)* et à l'objet *?nom-obj*.

REMARQUE : Aucune précondition n'est nécessaire ici. Elles sont en effet toutes contenues dans *aller-en* et *lâcher-objet*.

4.6.5. Fonction *monter-sur-objet (?nom-obj)*

C'est la fonction par laquelle le singe grimpe sur un objet.

Préconditions :

- *?nom-obj* est un nom d'objet physique
- le singe est sur le sol
- l'objet *?nom-obj* est sur le sol
- la position horizontale de *?nom-obj* est la même que celle du singe
- le singe ne porte rien

Postconditions :

- le singe est sur l'objet *?nom-obj*

Définition KIF :

```
(DEFFUNCTION monter-sur-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (pos-ver (état-du-singe ?état)) 'sol)
             (= (pos-ver (objet-de-nom ?nom-obj)) 'sol)
             (= (pos-hor (objet-de-nom ?nom-obj)) (pos-hor (état-du-singe ?état)))
             (= (objet-porté (état-du-singe ?état)) 'rien))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        ?nom-obj
                        (objet-porté (état-du-singe ?état)))
                (état-des-objets-physiques ?état))))
```

En français : appliquée aux arguments *?état* et *?nom-obj*, et si

- *?état* représente un état du problème ET
- *?nom-obj* est un nom d'objet physique ET
- la position verticale de la représentation du singe dans *?état* est "sol" ET
- la position verticale de l'objet *?nom-obj* est "sol" ET
- la position horizontale de *?nom-obj* et de la représentation du singe dans *?état* est la même ET
- le singe ne porte rien

alors la fonction monter-sur renvoie un nouvel état dans lequel la position verticale du singe est *?nom-obj*.

4.6.6. Fonction *sauter-sur-sol*

C'est la fonction par laquelle le singe, monté sur un objet, saute de celui-ci et tombe sur le sol. Cette fonction n'a pas d'autre argument que l'état courant du problème.

Préconditions :

- le singe n'est pas sur le sol

Postconditions :

- le singe est sur le sol

Définition KIF :

```
(DEFFUNCTION sauter-sur-sol (?état) :=
    (IF (AND (est-un-état-du-problème ?état)
             (/= (pos-ver (état-du-singe ?état)) 'sol))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        'sol
                        (objet-porté (état-du-singe ?état)))
                (état-des-objets-physiques ?état))))
```

En français : appliquée à l'argument *?état* et si

- ?état représente un état du problème ET
- la position verticale de la représentation du singe dans *?état* n'est pas "sol"

alors la fonction *sauter-sur-sol* génère un nouvel état dans lequel la position verticale du singe est "sol".

4.6.7. Remarque sur le non-respect des préconditions

Toutes les fonctions décrites ci-dessus ont la forme globale suivante :

(DEFFUNCTION fonction (@arguments) :=
(IF (<préconditions>) (<terme résultant>)))

Appliquer une fonction dont les préconditions ne sont pas respectées produit ⊥ comme résultat puisqu'il n'y a pas de terme "alternatif" (voir 3.3.1. la définition d'un terme conditionnel) qui donne une valeur au résultat de la fonction si les préconditions ne sont pas respectées.

L'objet ⊥ ne vérifie pas la relation *est-un-état-du-problème*. Dès lors, toute application d'une autre fonction à ⊥, résultat de la fonction dont les préconditions ne sont pas respectées, ne peut lui aussi que produire ⊥ puisqu'au moins une des préconditions (*est-un-état-du-problème*) n'est pas vérifiée. En anticipant un peu sur la suite de ce chapitre, nous pouvons dire que la solution du problème est une composition de fonctions de changement d'état. Si l'une de ces fonctions n'avait pas ses préconditions vérifiées, son résultat serait ⊥, donc le résultat de la fonction suivante dans la composition serait ⊥ et ainsi de suite... Une telle solution ne pourrait jamais générer un état final qui serait élément de l'ensemble des états-buts. On peut en déduire que l'expression d'une solution ne contiendra que des fonctions dont les préconditions sont vérifiées.

4.7. EXPRESSION DE LA SOLUTION

4.7.1 Séquence d'actions

La solution que le système doit générer a la forme d'une séquence d'actions, actions qui correspondent aux fonctions de changement d'état. Son expression la plus évidente en KIF est donc une liste d'actions.

Chaque action a la forme suivante : [nom de l'action, paramètres]. Ces paramètres sont les arguments de la fonction correspondante **à l'exception** de l'argument indiquant l'état. Il est inutile (et même redondant) de l'indiquer. En effet, l'état sur lequel s'applique la première action de la séquence est l'état initial du problème ; l'état sur lequel s'applique l'action (i) de la séquence (i >1) est celui produit par l'application de l'action (i-1).

La partie "paramètres" de la description de l'action est donc une liste d'arguments de fonction.

Pour que l'expression de cette solution soit tout à fait valide (au moins en tant que séquence d'actions du problème), il faut encore que chacune des actions corresponde biunivoquement à une fonction de changement état, et que les paramètres de cette action soient des arguments applicables par la fonction correspondante. Rappelons-nous qu'une fonction en KIF est définie comme un ensemble de listes dont les n-1 premiers items sont les arguments et le n-ième est le résultat de la fonction (voir 3.2.2.). Les paramètres de l'action sont donc les 2 à

(n-1)èmes items de la liste correspondante dans la fonction (le premier item étant omis, puisqu'il s'agit de l'état sur lequel la fonction s'applique).

Nous pouvons donc définir une relation exprimant qu'un objet KIF représente une séquence d'actions du problème de la manière suivante :

```
(DEFRELATION est-une-séquence-d-actions (?seq) :=
    (AND (list ?seq)
         (FORALL (?action)
                 (=> (item ?action ?seq)
                     (AND (double ?action)
                          (est-une-fonction-du-problème (first ?action))
                          (EXISTS (?élément-de-fonction)
                                  (AND (member ?élément-de-fonction (first ?action))
                                       (= (rest (butlast ?élément-de-fonction))
                                          (last ?action)))))))))

(DEFRELATION est-une-fonction-du-problème (?fonction) :=
    (member ?fonction (SETOF saisir-objet lâcher-objet aller-en
                             mettre-objet-en monter-sur-objet sauter-sur-sol)))
```

En français : un objet KIF représente une séquence d'actions si et seulement si

- il est une liste
- pour chaque item de cette liste :
    - cet item est une liste de deux composantes
    - la première de ces composantes est une fonction du problème (elle appartient à l'ensemble des fonctions du problème)
    - la seconde est identique à une liste d'arguments applicables par la fonction (c'est-à-dire que la définition en extension de la fonction vue comme un ensemble de listes, contient une liste composée de cette liste d'arguments et d'un résultat), liste d'arguments dont on a ôté le premier item

Un objet KIF respectant la relation *est-une-séquence-d-actions* décrit donc une séquence quelconque d'actions parmi les actions décrites sur le système. Il faut remarquer que la relation *est-une-séquence-d-actions* n'impose rien de plus à l'objet KIF en question. Une séquence composée d'actions dont les préconditions ne sont jamais respectées vérifie cette relation. En quelque sorte, la relation *est-une-séquence-d-actions* ne s'occupe que de la syntaxe d'une liste d'actions, pas de sa sémantique.

4.7.2. Exécution d'une séquence d'actions

Pour traiter cette sémantique d'une séquence d'actions, il est nécessaire de définir une fonction qui calcule le résultat de l'application de cette séquence à un état de départ. C'est la fonction *exécuter-séquence* dont voici la définition KIF :

```
(DEFFUNCTION exécuter-séquence (?seq ?état-de-départ) :=
    (IF (AND (est-une-séquence-d-actions ?seq)
             (est-un-état-du-problème ?état-de-départ))
        (IF (null ?seq)
            ?état-de-départ
            (exécuter-séquence (rest ?seq)
                               (apply (first (first ?seq))
                                      (append (LISTOF ?état-de-départ)
                                              (last (first ?seq))))))))
```

En français : appliquée aux arguments *?seq* et *?état-de-départ*, et si *?seq* représente une séquence d'actions et *?état-de-départ* un état du problème, la fonction *exécuter-séquence* génère un état du problème[26] qui est :

- si la séquence est vide, l'état de départ lui-même
- sinon, l'état généré par la fonction *exécuter-séquence* appliquée à la séquence dont on a ôté le premier item, et à l'état généré par l'application de la fonction décrite par ce premier item à l'état de départ.

REMARQUE : Cette définition, comme la plupart des définitions de fonction sur des listes, est récursive.

4.7.3. Définition de la solution

Rappelons-nous qu'à la fin de la section 5, nous avons défini la relation *est-état-solution-du-problème*. Un état vérifie cette relation si et seulement s'il est un état-but du problème, c'est-à-dire un état dans lequel le singe tient l'objet-cible.

Une *solution* du problème est une séquence d'actions qui, à partir de l'état initial donné, produit un état qui vérifie la relation *est-état-solution-du-problème*.

Il est évident que cette solution n'est pas unique. Nous en donnerons donc une définition sous la forme d'une relation KIF. Voici cette définition :

```
(DEFRELATION est-solution-du-problème (?seq) :=
    (est-état-solution-du-problème (exécuter-séquence ?seq état-initial-du-problème)))
```

En français : un objet KIF représente une solution du problème si et seulement si l'état généré par l'exécution de cette séquence à partir de l'état initial donné est un état-solution du problème.

L'expression de la solution que nous venons de proposer, à travers les définitions de fonction et de relation *est-une-séquence-d-actions*, *exécuter-séquence* et *est-solution-du-problème*, a l'avantage d'être indépendante du contexte. Pour peu que les relations *est-une-fonction-du-problème* et *est-état-solution-du-problème*, qui sont dépendantes du problème posé, soient correctement définies par le spécifieur, notre expression de la solution peut s'appliquer à tout problème de planification.

REMARQUE 1 : Une variante possible est de définir un "ensemble des solutions du problème" qui reprend toutes les séquences qui sont solution du problème, et définir la relation "est solution" comme étant celle que vérifient tous les éléments de cet ensemble.

REMARQUE 2 : Cette définition n'inclut aucune notion de solution optimale. Elle ne fait aucune différence entre une solution "droit-au-but" et une solution qui repasserait plusieurs fois par le même état. La notion d'optimalité est, dans ce

---

[26] ou, bien sûr, $\perp$ si une précondition à un moment donne n'est pas respectée.

problème, plutôt vague. Une solution optimale peut être celle qui minimise le nombre d'actions dans la séquence ; une autre peut être basée sur une pondération des différentes actions suivant leur niveau de difficulté physique pour le singe, etc. Dans le chapitre 6, nous envisagerons la notion de solution optimale comme étant la solution produite en appliquant une stratégie "orientée-buts". Une telle solution va en principe droit au but.

REMARQUE 3 : Un aspect important dans la recherche d'une solution est justement cette stratégie de résolution, qui fait partie de ce qu'on appelle la "métaconnaissance" du problème. Nous aborderons cet aspect par une recherche personnelle, que nous présenterons séparément au chapitre 6. On peut quand même remarquer que le nombre de stratégies applicables par un système expert est limité, et que ces stratégies devraient être une fois pour toutes formalisées dans les langages comme KIF, et représentées par des mots réservés comme "goal-oriented (strategy)". Bien entendu, certains aspects d'une stratégie restent dépendants du type de problème, et doivent donc toujours être spécifiés.

4.8. TRANSFORMATION D'ALBERT EN KIF

Cette section opère le lien entre l'analyse des besoins du problème du singe écrite en ALBERT au chapitre 2, et la spécification abstraite en KIF que nous venons d'élaborer. Elle propose une esquisse de démarche transformationnelle d'ALBERT à KIF, en mettant en évidence la correspondance des concepts spécifiés dans l'un et l'autre langage.

4.8.1. Le problème majeur de transformation

Il est clair que deux langages conçus dans des optiques aussi différentes que celle d'ALBERT (représentation des besoins d'un système) et celle de KIF (représentation abstraite des connaissances) ne sont pas *a priori* faits l'un pour l'autre et que certaines transformations de l'un à l'autre sont difficiles à opérer de façon directe.

La grande différence entre ALBERT et KIF est le type de spécification. ALBERT est fondamentalement un langage orienté-objet, tandis que KIF est plutôt fonctionnel (au sens LISP du terme). Cela pose un problème de représentation d'une part, et de démarche d'autre part. En ce qui concerne la représentation, nous verrons au point suivant comment les différents agents identifiés en ALBERT sont modélisés en KIF. Quant à la démarche de spécification en ALBERT, elle est l'inverse de la démarche KIF : ALBERT transforme une spécification d'agents complexes (décomposables) en une spécification d'agents terminaux (ceux qui exécutent ou subissent réellement les actions). A l'inverse, la démarche que nous avons proposée en KIF identifie d'abord les acteurs du problème[27] (les agents d'ALBERT) puis les regroupe en une seule composante (dans notre cas, sous forme d'une liste composée du singe et de l'ensemble des objets physiques).

Par ailleurs, les actions du problème sont vues en KIF comme des fonctions de changement d'état (état global du problème), et sont spécifiées globalement, alors

[27]Plus exactement, elle définit les *états* possibles de ces acteurs. Chacun de ces états peut donc être une représentation de l'acteur à un moment donné.

qu'en ALBERT, elles sont éclatées entre les différents agents qui ne voient de ces actions que les effets qu'elles ont sur eux-mêmes, ainsi que les conditions auxquelles elles peuvent s'exécuter, dans la mesure où ces conditions les impliquent.

Pour ces raisons, la démarche de transformation d'ALBERT en KIF partira des spécifications terminales d'ALBERT pour remonter vers les spécifications initiales (vers les agents de plus en plus complexes). Cette démarche nous permettra d'identifier les agents terminaux, puis de les agréger en agents complexes, jusqu'à spécifier l'agent complexe du niveau le plus haut, qui décrit l'entièreté du système ("chambre", dans notre cas).

4.8.2. Les agents du problème

Typage des agents

La spécification terminale en ALBERT nous montre deux types d'agents : le singe (individu) et les objets physiques (population). Dans la spécification de l'agent "chambre", ces agents sont des attributs typés. Les types de ces attributs sont définis comme des types complexes (voir 2.3.2., "complex type declarations"). Comme le typage n'existe pas en KIF, la correspondance entre les types complexes d'ALBERT et la spécification KIF peut être assurée par la définition des objets KIF *produit-cartésien-des-objets-physiques* et *produit-cartésien-du-singe*. Ces objets KIF représentent en fait des ensembles de listes, chaque liste étant une représentation *a priori* correcte (du point de vue du type) d'un état de l'agent concerné. Par exemple, la liste *[[5,10], 'sol, 'échelle]*, qui appartient à l'ensemble *produit-cartésien-du-singe*, représente un état du singe qui peut correspondre au type complexe SINGE défini en ALBERT[28]. Rappelons toutefois que les listes appartenant à ces ensembles "produit cartésien" ne sont que des représentations correctes au niveau du typage, sans plus. Ce qui définit réellement une représentation correcte, c'est l'ensemble des invariants du problème.

Représentation des agents du problème

Puisqu'on ne peut pas, en KIF, représenter le concept d'*objet* (dans le sens O-O du terme, c'est-à-dire un objet pouvant prendre des états successifs différents), on décrira les états de ces objets (ou agents, en ALBERT) sous forme de relations KIF, c'est-à-dire de prédicats exprimant qu'une représentation quelconque d'objet est bien une représentation possible de l'objet désigné, qui respecte les invariants définis pour cet objet. Ces invariants sont ceux qu'on retrouve dans le diagramme de contraintes associé à l'objet (agent) en question en ALBERT.

Dans le cas du singe, la relation *est-le-singe (?singe)* exprime que l'objet KIF désigné par *?singe* est une représentation possible du singe, qui correspond au type SINGE et qui vérifie les invariants du singe.

En ce qui concerne les objets physiques, qui sont une population, il est intéressant de les représenter comme un ensemble (ce que nous avons fait en ALBERT dans la spécification de l'agent "chambre"), afin de raisonner sur les propriétés de cet ensemble (complétude, unicité, etc.). L'ensemble des objets

---

[28]Notons l'immédiate correspondance entre les listes de KIF et le constructeur de type "produit cartésien". Une liste KIF dont le nombre d'éléments est connu peut représenter un produit cartésien.

physiques est donc un sous-ensemble de leur produit cartésien, qui respecte les invariants des objets physiques, aussi bien ceux relatifs à un objet physique indépendamment des autres ("un objet ne peut être posé sur lui-même") que ceux relatifs à l'ensemble des objets physiques pris comme un tout. La relation *est-l-ensemble-des-objets-physiques (?ens)* exprime que l'objet KIF désigné par *?ens* est une représentation possible de l'ensemble des objets physiques, ensemble dont chaque élément correspond bien au type OBJET défini en ALBERT, et dans lequel les invariants des objets physiques sont respectés. La relation *est-un-objet-physique (?obj)* exprime que l'objet KIF désigné par *?obj* est une représentation possible d'un objet physique.

Enfin, l'agent "chambre", qui regroupe l'agent "singe" et l'ensemble des agents "objet" est vu en KIF comme une liste de deux items : le premier est une représentation possible du singe, le deuxième est une représentation possible de l'ensemble des objets physiques. De plus, ces deux représentations doivent encore vérifier certains invariants qui les mettent en relation (p.ex., "un objet porté par le singe est à la même position horizontale que celui-ci").

REMARQUE : le cas de l'objet cible est un peu à part, car il s'agit d'une notion dépendante du contexte. Nous avons décidé d'exprimer cette notion de manière indépendante en KIF (voir 4.5.), alors qu'en ALBERT il est vu comme un attribut du singe dans la spécification terminale.

Esquisse de démarche de transformation des agents

Suite à la réflexion proposée ci-dessus, tentons de dégager une démarche plus générale de transformation des agents ALBERT en KIF.

1. ***Identifier les agents terminaux.*** Se baser sur la spécification terminale en ALBERT.
2. ***Définir un produit cartésien pour chacun de ces agents.*** Ce produit cartésien est un ensemble de listes, et les items de ces listes doivent correspondre aux types de chacun des attributs de l'agent terminal. L'ensemble des listes doit recouvrir toutes les représentations de l'agent qui sont correctes du point de vue du typage.
3. ***Identifier les invariants de chaque agent terminal.*** Dans le cas d'un agent "individu", ces invariants sont entièrement décrits dans la spécification terminale. Dans le cas d'un agent "population", ils sont décrits à la fois dans la spécification terminale (invariants propres à un agent indépendamment des autres) et dans la spécification non terminale supérieure (invariants relatifs à l'ensemble de ces agents).
4. ***Traduire ces invariants en syntaxe KIF.***
5. ***Pour chaque agent individuel X, définir une relation KIF "est-X (?x)".*** Cette relation est vérifiée si *?x* est un élément du produit cartésien correspondant à X, et si les invariants de X sont vérifiés (voir par exemple la définition de la relation *est-le-singe (?singe)*).
6. ***Pour chaque population d'agents Y, définir une relation KIF "est-l'ensemble-des-Y (?ens-y)".*** Cette relation est vérifiée si *?ens-y* est un sous-ensemble du produit cartésien correspondant à Y, et si les invariants de Y sont vérifiés (voir par exemple la définition de la relation *est-l-ensemble-des-objets-physiques (?ens-obj)*).

7. ***Pour chaque agent complexe, identifier les agents qui le composent.*** Cette composition est décrite dans le diagramme de déclaration de l'agent complexe (voir chapitre 2, figure 2.3.).
8. ***Pour chaque agent complexe Z, définir une relation KIF "est-Z (?z)" ou "est-l'ensemble-des-Z (?ens-z)".*** Si Z est un agent individuel au niveau supérieur (ou qu'il n'y a pas de niveau supérieur), définir "est-Z". Si Z fait partie d'une population, définir "est-l'ensemble-des-Z". Ces relations sont vérifiées si d'une part, *?z* (ou chaque élément de *?ens-z*) est une liste dont les différents items représentent les agents qui le composent, et si d'autre part les invariants relatifs à cet agent complexe (ou à cet ensemble d'agents complexes) sont vérifiés. En ALBERT, ces invariants sont décrits au niveau de spécification où l'agent complexe est encore décrit comme un agent terminal, et au niveau immédiatement supérieur si Z fait partie d'une population d'agents (voir par exemple la définition de la relation *est-un-état-du-problème (?état)*, qu'on aurait pu appeler *est-la-chambre (?chambre)*).
9. ***Réitérer les points 7 et 8 au niveau supérieur.*** Le processus s'arrête lorsqu'on a défini une relation pour l'agent complexe du niveau le plus haut, celui qui représente l'entièreté du système.

Si nous avions appliqué cette démarche de transformation au problème du singe, nous aurions obtenu pratiquement la même spécification KIF que celle que nous avons élaborée dans ce chapitre, à ceci près que l'objet cible aurait été décrit comme une composante du singe, et non comme un concept à part entière.

4.8.3. Les actions du problème

En KIF, les actions d'ALBERT sont décrites sous forme de fonctions de changement d'état. A partir d'un état donné du problème, elles génèrent un autre état. Ces états du problème correspondent à des états de l'agent complexe du niveau le plus haut, en l'occurrence l'agent "chambre". Remarquons qu'en KIF, l'état de départ doit être un paramètre de la fonction.

Expression des préconditions

Les préconditions d'une action sont décrites[29] dans les clauses *F* des responsabilités d'agent, dans le diagramme de contraintes associé à chacun des agents actifs (ceux qui ont l'initiative de l'action) . Une expression complète de ces préconditions est aussi contenue dans le diagramme de contraintes de l'agent complexe du niveau le plus haut (la chambre).

Une fonction de changement d'état en KIF a la forme générale suivante :

(DEFFUNCTION f (?état @arguments) :=
(IF (<préconditions(f)>) (<terme résultant(f)>))

Pour transformer l'expression des préconditions d'une action en ALBERT en une description en KIF, il suffit de :

1. prendre la négation des propositions contenues dans les clauses *F* relatives à l'action concernée dans le diagramme de contraintes associé à l'agent qui a l'initiative de cette action ;

[29]Plus exactement, c'est leur négation qui est décrite dans ces clauses.

2. ajouter une précondition "standard" relative à l'état global du problème qui dit que l'état passé en paramètre est bien un état cohérent du problème (c'est-à-dire qu'il donne une représentation correcte de l'agent complexe du niveau le plus haut). Dans notre spécification KIF, cette précondition est *(est-un-état-du-problème ?état)*.
3. relier ces préconditions par un opérateur logique KIF *AND*.

Etat résultant d'une action

En ALBERT, les effets des actions sur l'état d'un agent sont décrits dans le diagramme de contraintes associé à cet agent. Ils indiquent les changements de valeur des attributs de l'agent.

Comme on vient de le voir, chaque agent est représenté dans la description de l'agent complexe dont il fait partie, et celui-ci est à son tour représenté dans la description de l'agent complexe du niveau supérieur dont il fait partie, etc. Par transitivité, chaque agent terminal est donc décrit comme une composante de l'agent complexe du niveau le plus haut. S'il est individuel, il est représenté sous la forme d'une liste composée de ses attributs ; s'il fait partie d'une population, c'est cette population qui est représentée comme un ensemble de listes, chaque liste correspondant à une instance de l'agent. Dès lors, l'état résultant de la fonction de changement d'état peut être décrit à partir de l'état de départ (qui est un état de l'agent complexe du niveau le plus haut), en modifiant les composantes de cet état suivant les effets de l'action sur les agents correspondant à ces composantes, en utilisant les opérateurs définis en KIF sur les listes, les ensembles, etc.

Dans le cas de l'action *saisir-objet*, par exemple, l'état résultant peut se déduire des effets de l'action décrits pour le singe (objet-porté = o) et pour l'objet concerné (o.pos-ver = porté). Cet état s'obtient en modifiant la composante "singe" de l'état de départ (qui reste une liste dont l'item "objet-porté" prend une nouvelle valeur) et la composante "ensemble des objets physiques" (le nouvel ensemble correspond à l'ancien dont on a ôté l'élément représentant l'objet saisi par le singe, et auquel on a rajouté une nouvelle représentation de cet objet, dans laquelle l'attribut de position verticale est "porté").

Démarche de transformation des actions

La démarche globale de transformation d'une action ALBERT en fonction de changement d'état KIF consiste donc à :

1. Exprimer les préconditions à partir des clauses F associées à l'agent qui a l'initiative de l'action du niveau (voir ci-dessus)
2. Décrire l'état résultant par rapport à l'état de départ en identifiant les différentes composantes de cet état et en exprimant par des fonctions standards KIF les modifications de ces composantes. Ces modifications sont décrites dans le diagramme de contraintes de la spécification du niveau le plus haut, dans la rubrique "effects of actions". S'il y a un effet de bord, comme c'est le cas de la fonction "Aller-en", l'expression de l'état résultant peut elle-même être un terme conditionnel dans lequel on distingue les cas possibles (pas d'effet / effet) et les différents états produits en fonction de ces cas.
3. Donner à la fonction de changement d'état KIF sa forme standard (voir page précédente) en incluant le paramètre représentant l'état de départ, en décrivant

les préconditions sous la forme d'une proposition logique composée avec l'opérateur *AND*, et en décrivant l'état résultant comme terme résultant de la fonction.

4.8.4. Divers

Contraintes temporelles

Les contraintes temporelles ne sont pas descriptibles en KIF. Elles sont toutefois dérivables du texte KIF. Les contraintes impliquant l'opérateur □ ("toujours vrai dans le futur") sont vérifiées si l'on montre que chaque fonction de changement d'état produit un état dans lequel la proposition à laquelle s'applique cet opérateur est vérifiée. Les contraintes impliquant l'opérateur ◊ ("sera vrai à un moment donné dans le futur") sont vérifiées si l'on montre qu'il existe une composition de fonctions de changement d'état telle que l'état final résultant de cette composition vérifie bien la proposition à laquelle cet opérateur s'applique. On peut appliquer le même type de raisonnement aux autres opérateurs temporels.

Règles de causalité et clauses O et X

L'expression de clauses indiquant l'obligation pour une action de se produire si une autre action s'est produite n'est pas possible en KIF. D'autre part, si le sens des clauses O et X indique une obligation pour un agent actif d'exécuter une action lorsqu'une certaine situation est rencontrée, alors ces clauses ne sont pas non plus exprimables en KIF. Ces clauses retourneront donc au niveau informel dans le document de spécification abstraite de la solution.

Autres éléments d'une spécification KIF

Il est clair que le texte KIF dérivé d'une spécification en ALBERT ne met en évidence que les agents du problème et les actions possibles dans le système. Les autres éléments que nous avons présenté (but du problème, état initial, expression de la solution), qui sont plus proches de l'idée de "solution" du problème, viennent directement de la spécification informelle.

CHAPITRE 5
EVALUATION DE KIF

5.1. CRITERES D'EVALUATION

Le but de ce chapitre est d'évaluer le langage KIF en tant que langage de spécification abstraite des systèmes de connaissances. Nous avons déjà dit que KIF n'est pas conçu au départ comme langage de spécification formelle à partir d'un texte informel, mais plutôt comme langage d'abstraction d'un texte formel dans un langage donné. Notre évaluation ne sera donc pas une critique du langage en tant que tel, mais bien une évaluation des concepts présents en KIF dans le cadre d'une utilisation comme langage de spécification abstraite et formelle d'un système expert.

Pour évaluer un langage de spécification abstraite, il faut le soumettre à la pratique, c'est-à-dire l'expérimenter sur plusieurs problèmes de spécification différents. A partir de ces différentes spécifications, et dans le cadre de la spécification d'un système de connaissances, on peut appliquer les critères d'évaluation suivants :

1. **Le langage permet-il d'exprimer tous les concepts qui demandent une spécification formelle?** Est-il possible de décrire tous les objets du problème et les différentes interactions et relations entre eux? Est-il possible d'énoncer des vérités (invariants, par exemple) sur le monde du problème? Est-il possible de représenter la solution du problème? Est-il possible de représenter la stratégie à appliquer?
2. **Le langage permet-il de vérifier certaines propriétés de la spécification formelle?** La spécification est-elle complète? Peut-on vérifier la consistance de certaines définitions d'objets du problème? Peut-on vérifier la correction des éléments de la spécification? Peut-on prouver l'existence d'une solution? Si l'on dispose d'une spécification formelle du même problème dans un autre langage de spécification, peut-on prouver l'équivalence des deux spécifications?
3. **Quel est le degré d'adéquation du langage à la spécification informelle d'un problème?**
4. **Le langage est-il facile à manipuler?** Est-il lisible et facile à écrire? Peut-on typer les données? Dispose-t-on d'une librairie de fonctions de base? Etc.

Dans notre cas, nous avons opéré une seule mise en pratique et nous devrons nous contenter de celle-ci pour évaluer KIF en fonction des critères 1 à 4 ci-dessus.

L'évaluation de KIF par rapport au premier critère se fait directement à partir du texte de la spécification présenté au chapitre précédent. Nous y reviendrons dans la section "Evaluation globale de KIF", sous le titre "Spécification du problème". Rappelons que la question de la stratégie fait l'objet d'un chapitre à part (le chapitre 6).

Le deuxième critère est très vaste, et il ne nous sera pas possible d'envisager la vérification de toutes les propriétés de la spécification, principalement parce que

nous ne disposons pas d'une spécification du même problème dans un autre langage formel[30], et que nous ne pouvons dès lors vérifier que des propriétés intrinsèques à la spécification. Notre évaluation de KIF en fonction de ce critère se limitera donc à deux points :

- *Vérification de la consistance des fonctions de changement d'état*, c'est-à-dire du fait qu'elles produisent un nouvel état cohérent du système.
- *Vérification de l'existence d'au moins une solution*, indépendamment de la configuration initiale du problème.

Nous ferons la démonstration de ces propriétés dans les deux sections suivantes et évaluerons KIF par rapport à ce critère dans la section "Evaluation globale de KIF" sous le titre "Correction de la spécification". Une troisième démonstration relative à la stratégie sera effectuée au chapitre 6. Elle viendra étayer et nuancer notre évaluation de KIF.

Nous consacrerons une section à l'adéquation de la spécification KIF au texte informel afin de répondre au troisième critère d'évaluation.

Enfin, nous parlerons du critère de facilité d'utilisation de KIF dans la section "Evaluation globale de KIF" sous le titre "Divers", en nous basant sur notre propre expérience de manipulation du langage.

Cette évaluation revêt un caractère provisoire puisque KIF est toujours en évolution et que tous les outils qui le supportent ne sont pas encore pleinement développés.

5.2. PREUVE DE LA CONSISTANCE DES FONCTIONS DE CHANGEMENT D'ETAT

5.2.1. Principe

Il s'agit de montrer que les fonctions de changement d'état décrites au chapitre précédent sont consistantes au sein de la spécification, c'est-à-dire qu'une fonction dont les préconditions (en particulier, la cohérence de l'état de départ) sont vérifiées produit un état cohérent. En KIF, cette notion d'état cohérent est représentée par la relation *est-un-état-du-problème*, ce qui veut dire en pratique qu'un état [singe, objets physiques] généré par l'application d'une fonction de changement d'état doit vérifier cette relation.

Rappelons-nous cependant qu'une fonction de changement d'état ne produit pas forcément un état de type [singe, objets physiques]. Lorsque ses préconditions ne sont pas vérifiées, la fonction produit en effet $\perp$, qui n'est pas un état de ce genre et qui ne peut donc pas vérifier la relation *est-un-état-du-problème*. Il est clair qu'il ne faut démontrer la proposition *(est-un-état-du-problème (f ?état @args))* que lorsque les préconditions de f sont respectées.

---

[30]Bien qu'ALBERT soit plutôt un langage d'analyse des besoins qu'un langage de spécification comme Z ou VDM, on pourrait toutefois envisager une preuve formelle d'adéquation entre la représentation ALBERT et la représentation KIF du problème du singe, mais ceci dépasse quelque peu le cadre de notre exposé.

Toutes les définitions de fonction de changement d'état ont la forme KIF suivante :

(DEFFUNCTION f (?état @arguments) :=
(IF (<préconditions(f)>) (<terme résultant(f)>))

où le terme résultant a toujours la forme d'une liste de deux éléments représentant respectivement le singe et l'ensemble des objets physiques.

Le résultat de la fonction peut être :

- soit un état [singe, objets physiques] si les préconditions sont respectées. Cet état doit être cohérent.
- soit l'objet ⊥ si une des préconditions n'est pas respectée.

La consistance d'une fonction de changement d'état se traduit dès lors par la proposition logique suivante (écrite en KIF), que nous appellerons **proposition de consistance des fonctions de changement d'état** :

(FORALL (?f ?état @arguments)
(=> (est-une-fonction-du-problème ?f)
(/= ⊥ (apply ?f (LISTOF ?état @arguments)))
(est-un-état-du-problème (apply ?f (LISTOF?état @arguments)))))

Si cette proposition est vraie pour une fonction de changement d'état donnée f, alors f est consistante.

REMARQUE : Afin d'obtenir une spécification complète, nous pensons que la proposition de consistance des fonctions de changement d'état, et d'une façon générale toutes les propositions de consistance et de correction, doivent être intégrées au texte KIF, pour que ce texte puisse de lui-même permettre de montrer sa consistance.

On peut donc considérer que la proposition de consistance des fonctions de changement d'état est l'expression d'un théorème relatif à chacune de ces fonctions, dont les hypothèses (exprimées par le fait que la fonction ne produit pas le résultat ⊥) sont les préconditions de la fonction et la thèse, la proposition *est-un-état-du-problème (apply ?f (LISTOF ?état @arguments)))*.

Démontrer cette thèse revient en pratique à :

- d'une part, montrer que le résultat de la fonction est plausible, c'est-à-dire que les éléments qui composent ce résultat prennent leur valeur dans les valeurs possibles définies par les produits cartésiens correspondants (pas de poids d'objet physique autre que lourd ou léger, par exemple)
- d'autre part, montrer que tous les invariants sont conservés, aussi bien ceux relatifs à l'état que ceux relatifs à l'ensemble des objets physiques ou au singe.

On peut également opérer une deuxième découpe de la démonstration en traitant séparément le singe et les objets physiques. En appliquant ces deux découpes, nous pouvons déterminer une démarche pour démontrer la consistance d'une fonction de changement d'état f :

1. montrer que le nouvel état du singe est plausible <=> démontrer la proposition suivante :

(FORALL (?état @args)
(member (first (f ?état @args)) produit-cartésien-du-singe))

2. montrer que le nouvel état du singe conserve les invariants du singe <=> démontrer la proposition suivante :

(FORALL (?état @args)
(AND (invariant-singe-1 (first (f ?état @args)))...(invariant-singe-n (first (f ?état @args)))))

3. montrer que le nouvel état de l'ensemble des objets physiques est plausible <=> démontrer la proposition suivante :

(FORALL (?état @args) (subset (last (f ?état @args))produit-cartésien-des-objets-physiques))

4. montrer que le nouvel état de l'ensemble des objets physiques conserve les invariants de cet ensemble <=> démontrer la proposition suivante :

(FORALL (?état @args)
(AND (invariant-ens-obj-1 (last (f ?état @args)))...
(invariant-ens-obj-n (last (f ?état @args)))))

5. montrer que le nouvel état conserve les invariants de l'état <=> démontrer la proposition suivante :

(FORALL (?état @args)
(AND (invariant-état-1 (f ?état @args))...(invariant-état-n (f ?état @args))))

Chacune de ces démonstrations a pour hypothèse l'ensemble des préconditions de *f*. Dans les propositions à démontrer, les arguments de la fonction *(?état, @arguments)* sont quantifiés universellement.

Si l'on démontre les propositions 1 et 2, alors on démontre la proposition *(est-le-singe (first (f ?état @args)))*, par définition de la relation *est-le-singe*. De même, en démontrant les propositions 3 et 4, on démontre la proposition *(est-l-ensemble-des-objets-physiques (last (f ?état @args)))*, par définition de la relation *est-l-ensemble-des-objets-physiques*. Ayant démontré ces deux dernières propositions et la proposition 5, on peut en déduire la proposition *(est-un-état-du-problème (f ?état @args))* par définition de la relation *est-un-état-du-problème*. Il existe bien en effet un objet *?singe* - c'est *(first (f ?état @args))* - et un objet *?ens-obj* - c'est *(last (f ?état @args))* - tels que *(f ?état @args)* est une liste composée de ces deux items, qui vérifient respectivement *est-le-singe* et *est-l-ensemble-des-objets-physiques*; la proposition 5 établit la conservation des invariants sur l'état par *(f ?état @args)*.

Ayant démontré pour chaque fonction de changement d'état f, sous hypothèse que les préconditions de f sont vérifiées, la proposition *(est-un-état-du-problème (f ?état @args))*, on a démontré la proposition de consistance des fonctions de changement d'état, ce qui est notre but.

Montrons au travers d'un exemple qu'il est possible d'appliquer cette démonstration en cinq points[31].

5.2.2. Démonstration de la consistance d'une fonction

Pour simplifier, nous prendrons comme exemple la fonction *sauter-sur-sol* qui est la plus facile à exploiter, car son impact sur le système est mineur.

Les hypothèses dont nous disposons, à savoir les préconditions de la fonction, sont les deux suivantes[32] :

---

[31]Etant donné le caractère fastidieux d'une démonstration formelle, nous n'en ferons une que pour le premier point. Pour les suivants, nous nous limiterons à une courte preuve intuitive.

(est-un-état-du-problème ?état)

(/= (pos-ver (état-du-singe ?état)) 'sol)

Il faut démontrer la proposition suivante :

(est-un-état-du-problème (sauter-sur-sol ?état))

Ainsi qu'on vient de le voir, démontrer cette proposition est équivalent à démontrer les cinq points cités plus haut. Nous allons donc diviser la démonstration globale en cinq démonstrations auxiliaires.

1. Le nouvel état du singe est plausible

INTUITIVEMENT : le seul changement dans l'état du singe est sa position verticale, qui devient "sol".Celle-ci étant une valeur plausible pour la position verticale du singe, le nouvel état du singe reste plausible.

A démontrer :

(member (first (sauter-sur-sol ?état)) produit-cartésien-du-singe)) **(0)**

Suivant la définition de sauter-sur-sol, *first (sauter-sur-sol ?état)* vaut

LISTOF (pos-hor (état-du-singe ?état)) 'sol (objet-porté (état-du-singe ?état)) **(1)**

Cette liste est-elle bien une liste élément du produit cartésien du singe? Il s'agit bien d'une liste de trois items. Pour qu'elle appartienne au produit cartésien, il faut que chacun de ses items ait pour valeur l'une de celles qui lui sont imposées par la définition du produit cartésien du singe, c'est-à-dire que :

- le premier item doit être une liste [abs, ord] où abs et ord sont des entiers compris entre 1 et 10 ;
- le deuxième item doit appartenir à l'ensemble {sol, nom d'objet} où "nom d'objet" lui-même appartient à l'ensemble {bananes, lit, couverture, échelle} ;
- le troisième item doit appartenir à l'ensemble {rien, nom d'objet} où nom d'objet lui-même appartient à l'ensemble {bananes, lit, couverture, échelle}.

Puisque *?état* vérifie par hypothèse la relation *est-un-état-du-problème*, le terme *(état-du-singe ?état)*, obtenu par application de la fonction *état-du-singe*, est égal au premier item de *?état*, qui est une liste de deux items ; ce premier item vérifie la relation *est-le-singe* par définition de *est-un-état-du-problème* ; dès lors, la proposition

(member (état-du-singe ?état) produit-cartésien-du-singe) **(2)**

est vraie par définition de *est-le-singe*. Si on développe le terme *(état-du-singe ?état)* suivant les fonctions d'extraction *pos-hor*, *pos-ver* et *objet-porté*, on obtient la liste suivante :

```
(LISTOF (pos-hor (état-du-singe ?état))
        (pos-ver (état-du-singe ?état))
        (objet-porté (état-du-singe ?état)))
```

---

[32]Nous omettrons le quantificateur universel pour simplifier les écritures. Cette omission a un support théorique en KIF (voir [GENESERETH92] 3.0., §4.5).

qui est sémantiquement équivalente à *(état-du-singe ?état)* et vérifie donc la relation **(2)** ci-dessus. Le premier et le troisième item de cette liste sont respectivement égaux au premier et au troisième item de la liste **(1)**. Ceux-ci ont donc des valeurs plausibles. Reste à montrer que le deuxième item de cette liste a aussi une valeur plausible. Cet item vaut "sol" et "sol" est bien une des valeurs que, suivant la définition du produit cartésien du singe, ce deuxième item peut prendre, cette valeur est donc plausible.

Ayant démontré que les trois items de la liste **(1)** sont plausibles suivant la définition du produit cartésien du singe, nous avons montré que la liste **(1)** est bien un élément du produit cartésien du singe, la proposition **(0)** est donc démontrée, C.Q.F.D.

2. Les invariants du singe sont conservés

INTUITIVEMENT : nous avons vu qu'il n'y a pas d'invariants propres au singe, la liste d'invariants du singe est donc vide et il n'y a rien à démontrer à ce point.

3. Le nouvel état des objets physiques reste plausible

INTUITIVEMENT : On peut voir que la fonction sauter-sur-sol ne modifie pas l'état des objets physiques de départ. Celui-ci étant plausible parce qu'il est une composante de l'état de départ, qui est par hypothèse un état cohérent, le nouvel état des objets physiques (identique à l'ancien) reste plausible.

4. Les invariants de l'ensemble des objets physiques sont conservés

INTUITIVEMENT : Même raisonnement qu'au point 3. L'état des objets physiques étant inchangé, les invariants de l'ensemble des objets physiques sont conservés.

5. Les invariants de l'état global sont conservés

REMARQUE : Nous n'avons pas formalisé cet ensemble d'invariants. Il est évident qu'une spécification complète demande l'écriture en KIF de tous les invariants du problème, et que la preuve de consistance d'une fonction au sein d'une spécification ne peut être faite que si l'on démontre que tous ces invariants sont conservés.

En guise d'exemple, montrons intuitivement que l'invariant "si le singe tient un objet, cet objet est à la même position horizontale que lui" est conservé par la fonction *sauter-sur-sol*.

INTUITIVEMENT : la fonction sauter-sur-sol ne modifie qu'une seule valeur dans l'état du problème, la position verticale du singe. Dès lors, si le singe tient un objet dans l'état avant application de sauter-sur-sol, il le tient aussi dans l'état après. Par ailleurs, la position horizontale du singe et celle de cet objet restent inchangées. Puisqu'elles étaient égales avant application de la fonction, elles le sont toujours après.

Conclusion

Comme on l'a vu, si les cinq points ci-dessus sont démontrés pour toutes les fonctions de changement d'état, la proposition de consistance des fonctions de changement d'état est démontrée. Rappelons toutefois qu'il ne s'agit que de consistance et non de correction : on montre seulement que chaque fonction de

changement d'état produit bien, lorsque ses préconditions sont respectées, un nouvel état cohérent. Rien ne garantit que cette fonction produit l'état voulu par sa spécification informelle[33].

Nous pouvons donc conclure que la consistance des fonctions de changement d'état peut être montrée directement à partir de la spécification KIF du problème.

5.3. PREUVE DE L'EXISTENCE D'UNE SOLUTION

Dans cette section, nous allons voir qu'il est possible de montrer que le problème du singe et des bananes a au moins une solution UNIQUEMENT sur base des informations contenues dans le texte KIF et indépendamment de la configuration initiale (donc sans tenir compte de la partie dynamique du texte). Cette démonstration sera toutefois très dépendante du contexte du problème.

5.3.1. Thèse de la démonstration

Comme dans la démonstration précédente, la thèse de cette démonstration peut être exprimé par une proposition de consistance, la **proposition de consistance relative à l'existence d'une solution**. Voici cette proposition KIF :

```
(FORALL (?état-initial ?objet-cible)
        (=> (est-un-état-du-problème ?état-initial)
            (est-un-objet-physique ?objet-cible)
            (= (poids ?objet-cible) 'léger)
            (EXISTS (?seq)
                    (member
                        (exécuter-séquence ?seq ?état-initial)
                        (SETOFALL (?état)
                                  (AND (est-un-état-du-problème ?état)
                                       (= (objet-porté (état-du-singe?état))
                                          (nom ?objet-cible))))))))
```

En français : Pour tout état initial qui est un état du problème, pour tout objet-cible qui est un objet physique de poids léger (ce qui veut dire qu'état initial et objet-cible respectent les définitions partielles correspondantes dans la partie statique de la spécification), il existe au moins une séquence d'actions dont l'exécution à partir de l'état initial produit un état final qui appartient à l'ensemble des états qui sont états du problème et dans lesquels l'objet tenu par le singe est l'objet-cible.

Démontrer l'existence d'une solution au problème revient en principe à démontrer cette proposition de manière formelle. Pour éviter de devoir recourir à des expressions KIF complexes, nous allons ramener cette démonstration à un texte plus informel où nous montrerons qu'il est possible de déterminer une par une les actions à accomplir pour que le singe atteigne son but[34].

---

[33]On verra toutefois en fin de chapitre que certaines contraintes informelles peuvent être décrites formellement et donc permettre des preuves de correction des fonctions de changement d'état. Ces contraintes peuvent être décrites sous forme de proposition KIF et être insérées dans la spécification. Elles y jouent le même rôle de validation que les propositions de consistance.

[34]Pour le lecteur désireux de voir une démonstration formelle à partir du texte KIF, nous proposerons une telle démonstration dans le cadre du chapitre 6.

Nous diviserons cette démonstration en deux parties. D'abord, nous montrerons que, quelle que soit la configuration de départ, il est toujours possible de déterminer une séquence finie d'actions telle que l'état résultant de l'exécution de cette séquence est un état dans lequel le singe se trouve au sol, à un endroit donné (par exemple [5,5]) et ne porte rien. Ensuite nous montrerons qu'à partir d'un tel état, il est toujours possible de déterminer une séquence finie d'actions telle que l'état résultant de l'exécution de cette séquence est un état-but, quel que soit l'objet-cible désigné. La combinaison de ces deux séquences est une solution du problème.

5.3.2. Première séquence d'actions

REMARQUE PRELIMINAIRE 1 : Nous utiliserons une notation syntaxique plus simple que celle de KIF, où les listes seront représentées par [ ], les ensembles par { }. La correspondance entre cette syntaxe et celle de KIF est immédiate. Par ailleurs, certains des éléments descriptifs d'un état seront désignés par des variables. Cela ne veut pas dire que les valeurs courantes de ces éléments peuvent être quelconques. Elles dépendent en fait des valeurs possibles définies par le produit cartésien du singe et celui des objets physiques, et des invariants. Si nous désignons ces éléments par des variables, c'est qu'ils peuvent prendre différentes valeurs. En ce sens, ces variables sont des variables au sens traditionnel (et non KIF) du terme. Nous ferons également apparaître ces variables dans les expressions des fonctions KIF à appliquer. En agissant ainsi, nous réalisons une extension au langage en introduisant ce concept de variable au sens traditionnel, ce qui révèle une faiblesse de KIF (absence d'un concept) en tant que langage de spécification.

REMARQUE PRELIMINAIRE 2 : Nous considérerons que l'échelle est toujours accessible pour le singe, ce qui suppose que l'un des invariants de l'ensemble des objets physiques est que l'échelle ne peut pas être au plafond. Il est clair en effet que si l'objet-cible ainsi que l'échelle étaient tous deux au plafond, le singe ne pourrait jamais attraper cet objet-cible et il n'y aurait donc pas de solution. Pour la même raison, nous considérerons que le poids de l'échelle est obligatoirement "léger", ce qui est aussi exprimable par un invariant.

Pour fixer les esprits, voici les différentes possibilités d'action du singe :

Etape	Si oui, action	Résultat
1. Le singe porte un objet	• Lâcher-objet	• Le singe ne porte plus rien. Passer à l'étape 2
2. Le singe est sur un objet	• Sauter-sur-sol	• Le singe est au sol. Passer à l'étape 3
3. Le singe ne porte rien et est au sol	• Aller-en (5,5)	• Le singe est en (5,5)

Décrivons d'abord un état initial quelconque :

[[[x-singe, y-singe], pos-ver-singe, objet-porté-singe],
{['bananes', [x-bananes, y-bananes], pos-ver-bananes, poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

N'importe quel état initial (ou quelconque) peut être décrit par la représentation ci-dessus, en instantiant les variables décrites. Puisqu'un état initial est un état cohérent du problème, toutes les contraintes (domaines de valeurs, invariants) sont

respectées. Nous allons maintenant construire une séquence d'actions dont le résultat sera que le singe se trouvera en [5, 5], au sol, et ne tiendra aucun objet. Cette construction se fera fonction par fonction, et suivant les cas.

Etape 1 : le singe porte-t-il un objet?

SI OUI (objet-porté-singe <> "rien") , alors les préconditions de la fonction *(lâcher-objet (?état (objet-porté (état-du-singe ?état))))* sont respectées. En effet, ces préconditions sont les suivantes (suivant la définition KIF de la fonction) :

(est-un-état-du-problème ?état)

(est-un-objet-physique (objet-de-nom ?nom-obj))

(= (objet-porté (état-du-singe ?état)) ?nom-obj)

La preuve de la vérité de la première précondition est immédiate puisque l'état auquel on applique la fonction est un état initial qui, par définition, est un état cohérent du problème. Pour la même raison, on peut prouver la vérité de la deuxième précondition en instantiant *?nom-obj* par la valeur donnée ci-dessus, *objet-porté (état-du-singe ?état)* qui est le nom de l'objet porté par le singe dans l'état initial. Cet état étant un état cohérent du problème, l'objet porté par le singe est forcément un objet physique (puisque dans ce cas-ci il ne peut pas être "rien"), ce qui vérifie la deuxième précondition. La preuve de la vérité de la troisième précondition est immédiate en opérant la même instantiation : on obtient alors la proposition *(= (objet-porté (état-du-singe ?état)) (objet-porté (état-du-singe ?état)))* qui est évidemment vraie (x = x). C.Q.F.D.

L'application de cette fonction produit un état de la forme suivante, par définition de la fonction :

[[[x-singe,y-singe], pos-ver-singe, 'rien'],
{['bananes', [x-bananes, y-bananes], pos-ver-bananes, poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

dans lequel une des valeurs de position verticale (celle de l'objet que le singe portait) est devenue "sol". Puisque cet état est généré par une fonction de changement d'état et qu'on a prouvé à la section 1 que ces fonctions génèrent des états cohérents du problème, ce nouvel état est bien un état cohérent du problème.

SI NON (objet-porté-singe = "rien") l'état courant a déjà la forme ci-dessus, et on peut directement passer à l'étape 2. Notons que, l'état courant étant inchangé, il reste un état cohérent du problème.

Etape 2 : le singe est-il sur un objet?

SI OUI (pos-ver-singe <> "sol"), alors les préconditions de la fonction *(sauter-sur-sol (?état))* sont respectées (quoiqu'on ait fait à l'étape précédente, *?état* est un état cohérent du problème, comme on vient de le voir; les autres préconditions peuvent être vérifiées par des démonstrations du genre de celles que nous avons données à l'étape précédente). L'application de cette fonction produit un état de la forme suivante, par définition de la fonction :

[[[x-singe,y-singe], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], pos-ver-bananes, poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

SI NON (pos-ver-singe = "sol") l'état courant a déjà la forme ci-dessus, et on peut directement passer à l'étape 3.

Comme pour l'étape 1, l'état produit après application de cette étape est un état cohérent du problème.

Etape 3 : le singe doit aller en [5, 5]

Les préconditions de la fonction *(aller-en ?état 5 5)* sont respectées (à vérifier) et l'application de cette fonction produit un état de la forme suivante, par définition de la fonction :

[[[5, 5], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], pos-ver-bananes, poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

L'état ci-dessus est bien un état du problème dans lequel le singe se trouve en [5, 5], au sol, et où il ne porte rien.

Considérons maintenant l'ensemble des séquences d'actions dont les éléments sont toutes les listes composées à partir de la liste des 3 actions ci-dessus en ôtant un nombre quelconque (0 à 2) d'actions, n'importe lesquelles sauf la dernière (aller en [5, 5]), qui est obligatoire. Pour être clair, si nous appelons ces actions a, b et c, l'ensemble en question est le suivant : { [a, b, c], [a, c], [b, c],[c]}. On vient de montrer que, quel que soit l'état initial, il y a une de ces listes qui représente une séquence d'actions telle que son application produit un état dans lequel le singe est en [5, 5], au sol, et où il ne porte rien. C.Q.F.D.

5.3.3. Deuxième séquence d'actions

La démarche de raisonnement est similaire à celle de la première séquence d'actions, mais pas tout à fait identique. Nous partons ici d'un état donné du singe, dans lequel celui-ci est en [5, 5], au sol, et ne porte rien, et nous devons montrer qu'il est toujours possible de trouver une séquence d'actions telle que l'état final produit est un état-but, dans lequel le singe tient l'objet-cible. La démarche ne se décrit plus en étapes, mais en cas et sous-cas. Une première division en cas concerne la désignation de l'objet-cible. Celui-ci peut en effet être n'importe lequel des objets physiques de poids léger, suivant sa définition partielle[35]. Une deuxième division concerne la position verticale de l'objet-cible (au plafond ou ailleurs). En recoupant ces deux subdivisions, nous obtenons les cas suivants :

1. l'objet-cible est le régime de bananes, il est au plafond
2. l'objet-cible est le régime de bananes, il n'est pas au plafond

[35]Pour rappel, l'objet-cible n'est complètement défini qu'à la configuration initiale, et nous voulons ici montrer l'existence d'une solution indépendamment de cette configuration initiale.

3. l'objet-cible est le lit, il est au plafond
4. l'objet-cible est le lit, il n'est pas au plafond
5. l'objet-cible est la couverture, elle est au plafond
6. l'objet-cible est la couverture, elle n'est pas au plafond
7. l'objet-cible est l'échelle, elle n'est pas au plafond

Cette liste de cas couvre l'ensemble des possibilités de désignation et de position de l'objet-cible. Bien entendu, certains cas peuvent ne pas être possibles : si un des objets a un poids "lourd", il ne peut être l'objet-cible par définition de celui-ci. De même, il peut exister certains invariants qui empêchent le lit ou la couverture d'être au plafond, par exemple, ce qui rend impossible les cas 3 et 5.

Dans chacun de ces cas, nous montrerons qu'il existe une séquence d'actions permettant, à partir d'un état donné où le singe est en [5,5], au sol, et ne porte rien, de générer un état-but du problème, dans lequel le singe tient l'objet-cible. En pratique, nous nous limiterons aux cas 1 et 2, les autres cas étant similaires à l'un de ces deux-là (même type de démonstration, seul l'objet-cible change).

CAS 1 : L'objet-cible est le régime de bananes, il est au plafond

Dans ce cas, le singe doit exécuter les actions suivantes :

- **aller à la position de l'échelle**
- **prendre l'échelle**
- **mettre l'échelle sous les bananes**
- **monter sur l'échelle**
- **prendre les bananes**

L'état de départ est le suivant :

[[[5, 5], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

Les préconditions de la fonction *(aller-en ?état* x-éch y-éch*)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par la première séquence d'actions ;
- x-éch et y-éch sont bien des coordonnées de type entier comprises entre 1 et 10, puisqu'elles correspondent à la position horizontale de échelle dans l'état du problème, qui est cohérent ;
- la position verticale du singe est bien "sol".

On peut donc appliquer cette fonction qui ccorrespond à amener le singe l'endroit où se trouve l'échelle.

L'état généré par l'application de *(aller-en ?état* x-éch y-éch*)* est le suivant :

[[[x-éch, y-éch], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

Les préconditions de la fonction *(saisir-objet ?état 'échelle)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par l'application d'une fonction de changement d'état ;
- l'objet qui a pour nom *'échelle* est bien un objet physique respectant les invariants de l'ensemble des objets physiques, donc la relation *est-un-objet-physique* ;
- le poids de l'échelle est "léger" (ce qui est imposé par un invariant de l'ensemble des objets physiques) ;
- le singe et l'échelle sont à la même position horizontale [x-éch, y-éch] ;
- le singe est sur le sol et l'échelle n'est pas au plafond (ceci est imposé par un invariant de l'ensemble des objets physiques, comme on l'a vu plus haut).

L'application de cette fonction produit donc l'état suivant (selon la définition de la fonction) :

[[[x-éch, y-éch], 'sol', 'échelle'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], 'porté', poids-éch]}]

On peut appliquer la fonction *(mettre-objet-en ?état* x-bananes y-bananes*)* à condition que *(aller-en ?état* x-bananes y-bananes*)* soit applicable, et qu'on puisse appliquer *lâcher-objet* au nouvel état généré par *aller-en*[36].

Les préconditions de la fonction *(aller-en ?état* x-bananes y-bananes*)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré une fonction de changement d'état ;
- x-bananes et y-bananes sont bien des coordonnées de type entier comprises entre 1 et 10, puisqu'elles correspondent à la position horizontale des bananes dans l'état du problème, qui est cohérent ;
- la position verticale du singe est bien "sol".

L'état "intermédiaire" généré par cette fonction est le suivant :

[[[x-bananes, y-bananes], 'sol', 'échelle'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-bananes, y-bananes], 'porté', poids-éch]}]

[36]Pour rappel, la fonction *mettre-objet-en* est la composée des fonctions *aller-en* et *lâcher-objet*.

Les préconditions de la fonction *(lâcher-objet ?état 'échelle)*, où *?état* représente cet état intermédiaire, sont vérifiées. En effet :

- l'état intermédiaire est bien un état du problème puisqu'il a été généré par une fonction de changement d'état ;
- l'objet qui a pour nom *'échelle* est bien un objet physique respectant les invariants de l'ensemble des objets physiques, donc la relation *est-un-objet-physique* ;
- cet objet est bien porté par le singe dans l'état auquel on veut appliquer la fonction.

L'application de cette fonction, qui correspond à appliquer la fonction *(mettre-objet-en ?état 'échelle* x-bananes y-bananes*)* appliquée à l'état décrit au bas de la page précedente, et donc à mettre l'échelle sous les bananes, produit donc l'état suivant (selon la définition de la fonction) :

[[[x-bananes, y-bananes], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-bananes, y-bananes], 'sol', poids-éch]}]

Les préconditions de la fonction *(monter-sur ?état 'échelle)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par une fonction de changement d'état ;
- l'objet qui a pour nom *'échelle* est bien un objet physique respectant les invariants de l'ensemble des objets physiques, donc la relation *est-un-objet-physique* ;
- la position verticale du singe est bien "sol" ;
- la position verticale de l'objet qui a pour nom *'échelle* est bien "sol" ;
- le singe et l'objet "échelle" ont bien la même position horizontale qui est [x-bananes, y-bananes].

On peut donc appliquer cette fonction qui correspond à faire monter le singe sur l'échelle.

L'état généré par l'application de *(monter-sur ?état 'échelle)* est le suivant :

[[[x-bananes, y-bananes], 'échelle', 'rien'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-bananes, y-bananes], 'sol', poids-éch]}]

Les préconditions de la fonction *(saisir-objet ?état 'bananes)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par l'application d'une fonction de changement d'état ;

- l'objet qui a pour nom *'bananes* est bien un objet physique respectant les invariants de l'ensemble des objets physiques, donc la relation *est-un-objet-physique* ;
- le poids des bananes est "léger" par définition de l'objet-cible ;
- le singe et les bananes sont à la même position horizontale [x-bananes, y-bananes] ;
- le singe est sur l'échelle et les bananes sont au plafond.

On peut donc appliquer cette fonction qui correspond à faire prendre les bananes par le singe.

L'état généré par l'application de *(saisir-objet ?état 'bananes)* est le suivant :

[[[x-bananes, y-bananes], 'échelle', 'bananes'],
{['bananes', [x-bananes, y-bananes], 'porté', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-bananes, y-bananes], 'sol', poids-éch]}]

Cet état est bien un état solution du problème puisqu'il appartient à l'ensemble des états-buts. En effet :

- il s'agit bien d'un état du problème puisqu'il a été généré par une fonction de changement d'état ;
- dans cet état, l'objet porté par le singe ("bananes") est bien l'objet-cible.

C.Q.F.D.

CAS 2 : L'objet-cible est le régime de bananes, il n'est pas au plafond

Ce cas est plus simple que le précédent. Le singe ne doit exécuter que deux actions :

- **aller à la position des bananes**
- **prendre les bananes**

L'état de départ est le suivant :

[[[5, 5], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], 'plafond', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

Les préconditions de la fonction *(aller-en ?état* x-bananes y-bananes*)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par la première séquence d'actions ;
- x-bananes et y-bananes sont bien des coordonnées de type entier comprises entre 1 et 10, puisqu'elles correspondent à la position horizontale des bananes dans l'état du problème, qui est cohérent ;
- la position verticale du singe est bien "sol".

On peut donc appliquer cette fonction qui correspond à amener le singe à l'endroit où se trouvent les bananes.

L'état généré par l'application de (aller-en ?état x-bananes y-bananes) est le suivant :

[[[x-bananes, y-bananes], 'sol', 'rien'],
{['bananes', [x-bananes, y-bananes], pos-ver-bananes, poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

Les préconditions de la fonction *(saisir-objet ?état 'bananes)* sont vérifiées. En effet :

- *?état* est un état cohérent du problème, puisqu'il a été généré par l'application d'une fonction de changement d'état ;
- l'objet qui a pour nom *'bananes* est bien un objet physique respectant les invariants de l'ensemble des objets physiques, donc la relation *est-un-objet-physique* ;
- le poids des bananes est "léger" par définition de l'objet-cible ;
- le singe et les bananes sont à la même position horizontale [x-bananes, y-bananes] ;
- le singe est au sol et les bananes ne sont pas au plafond, par hypothèse.

On peut donc appliquer cette fonction qui correspond à faire prendre les bananes par le singe.

L'état généré par l'application de *(saisir-objet ?état 'bananes)* est le suivant :

[[[x-bananes, y-bananes], 'sol', 'bananes'],
{['bananes', [x-bananes, y-bananes], 'porté', poids-bananes],
['lit', [x-lit, y-lit], pos-ver-lit, poids-lit],
['couverture', [x-couv, y-couv], pos-ver-couv, poids-couv],
['échelle', [x-éch, y-éch], pos-ver-éch, poids-éch]}]

Cet état est bien un état solution du problème puisqu'il appartient à l'ensemble des états-buts. En effet :

- il s'agit bien d'un état du problème puisqu'il a été généré par une fonction de changement d'état ;
- dans cet état, l'objet porté par le singe ("bananes") est bien l'objet-cible.

C.Q.F.D.

Autres cas

On peut montrer qu'il est toujours possible de trouver une séquence d'actions partant d'un état initial dans lequel le singe est en [5, 5], au sol, et ne porte rien, et aboutissant à un état solution, dans lequel le singe tient l'objet-cible, quel que soit le cas parmi ceux que nous avons énuméré plus haut. Nous venons de démontrer les cas 1 et 2, dans lesquels l'objet-cible est le régime de bananes. Les cas 3 et 5 se démontrent de façon similaire au cas 1, en remplaçant "bananes" par l'objet-cible concerné. Les cas 4, 6 et 7 se démontrent de façon similaire au cas 2, en remplaçant également "bananes" par l'objet-cible concerné.

Tous les cas de figure sont donc démontrés.

5.3.4. Conclusion de la démonstration

Nous avons montré les deux points suivants :

1. quel que soit l'état initial du problème, il est toujours possible de construire une séquence d'actions dont l'état résultant est un état dans lequel le singe se trouve en [5, 5], au sol, et ne tient aucun objet.
2. à partir d'un tel état, il est toujours possible de construire une séquence d'actions dont l'état résultant est un état solution du problème.

Il est donc toujours possible de construire une séquence d'actions à partir de ces deux séquences (en ajoutant la deuxième à la première) qui, partant d'un état initial quelconque, produit comme résultat un état solution. Une telle séquence est par définition une solution du problème. Cette solution existe donc, C.Q.F.D.

REMARQUE : Nous avons démontré l'existence d'au moins une solution, mais pas l'unicité de cette solution, pour la simple raison qu'elle n'est jamais unique. Il serait intéressant de montrer qu'il est toujours possible de trouver une solution en un nombre borné d'actions.

5.4. ADEQUATION DE KIF A LA SPECIFICATION INFORMELLE DU PROBLÈME

Nous allons ici reprendre les contraintes relevées par J.-L. Bussenot, J. Foisseau et M. Lemoine dans leur exemple de modélisation informelle du problème [BUSSENOT92], et voir comment ces contraintes se retrouvent dans le texte KIF. Ayant jusqu'à présent ignoré l'aspect "stratégie" du problème, nous oublierons celles qui sont relatives à la représentation de cette stratégie (numérotées C21, C22 et C23 dans [BUSSENOT92]). Ensuite, nous montrerons le lien existant entre les définitions informelles des fonctions de changement d'état sous forme de pré- et postconditions, et leur définition formelle en KIF.

5.4.1. Représentation des contraintes informelles

C1. Il y a une seule échelle. L'échelle est un objet physique, appartenant à l'ensemble des objets physiques. En KIF, cet ensemble est notamment défini par ses invariants, en particulier celui que nous avons appelé "contrainte d'unicité", qui garantit qu'il n'y a qu'un seul objet de nom donné dans l'ensemble, ce qui est donc vrai pour l'échelle.

C2. Le singe peut se déplacer seul. Cette contrainte est ce qu'on peut appeler une "contrainte positive", qui à notre sens ne doit pas être représentée telle quelle dans un texte formel[37]. Dans ce cas particulier, on peut considérer que c'est cette spécificité du singe par rapport aux objets physiques qui nous a poussés à le traiter à part, comme un représentant unique de la classe "objet mobile".

C3. L'échelle est assez grande pour que le singe puisse toucher le plafond une fois dessus. Ca signifie qu'il est capable de saisir un objet au plafond,

[37]Les auteurs de la spécification informelle sont partagés à ce sujet.

au-dessus de lui, s'il est sur l'échelle. C3 est représentée comme précondition de la fonction *saisir-objet*.

C3bis. L'échelle est assez légère pour être portée par le singe. Son attribut "poids" est donc obligatoirement "léger", ce qui est représentable par un invariant sur l'ensemble des objets physiques.

C4. La taille des autres objets sera ignorée. Elle n'est effectivement pas présente dans la représentation des objets physiques en KIF.

C5. Le poids des objets physiques doit être représenté. Le poids est le dernier élément de la liste KIF qui représente un objet physique, et il peut prendre les valeurs *'lourd* ou *'léger*.

C6. Le singe voit tous les objets dans la pièce. C'est une contrainte positive (voir C2). Le concept de "champ de vision" du singe n'est pas représenté en KIF. Ce champ de vision est par défaut toute la pièce. Si on avait eu une contrainte négative à ce niveau ("le singe a un champ de vision limité"), il aurait fallu représenter ce champ de vision et rajouter des préconditions aux fonctions de changement d'état, par exemple à *aller-en (?x ?y)*, du genre "[x, y] est dans le champ de vision du singe".

C7. Le singe peut accomplir certaines actions sans qu'il faille lui dire en détail comment faire. Ce sont des actions macroscopiques. Elles correspondent aux fonctions de changement d'état que nous avons décrites en termes de pré- et postconditions, de façon fonctionnelle et non impérative. Il est possible que ces fonctions doivent être découpées de façon microscopique aux étapes de conception et d'implémentation du système.

C8. Le système produira une solution en termes des actions précédentes. Cette solution est représentée en KIF par une séquence d'actions sous la forme d'une liste, chaque action correspondant à une fonction de changement d'état.

C9. La localisation d'un objet est un attribut de l'objet. Elle est représentée par deux éléments de la liste représentant un objet physique, le deuxième (localisation horizontale) et le troisième (localisation verticale).

C10. La localisation horizontale est un point 2D. Elle est représentée en KIF par une liste de deux entiers.

C10bis. La valeur de la localisation verticale est sur-le-sol/au plafond/sur-objet(X) (ou "porté", voir C14.). En KIF, le produit cartésien des objets physiques définit la localisation verticale d'un objet comme appartenant à l'ensemble {sol, plafond, porté, nom d'objet} où "nom d'objet" représente l'objet sur lequel se trouve l'objet dont il est question. Même définition pour le singe, sauf que les valeurs " plafond" et "porté" n'appartiennent pas à l'ensemble des valeurs possibles de localisation verticale pour le singe.

C11. Tous les objets physiques ont la même représentation. En KIF, ils sont tous définis comme listes de même structure, appartenant à l'ensemble des objets physiques.

C11bis . L'objet-cible n'est pas distingué. Dans la partie statique de la spécification KIF, l'objet-cible est seulement partiellement défini comme étant un objet physique de poids léger. Il n'intervient pas dans les définitions relatives aux objets physiques.

C12. L'énoncé initial du problème précisera l'objet-cible. Cet énoncé initial est décrit en KIF par ce que nous avons appelé "partie dynamique de la

spécification". Celle-ci contient une définition partielle de l'objet-cible, qui, combinée à la définition partielle présente dans la partie statique, identifie complètement l'objet-cible.

C13. Le système modifiera la localisation horizontale d'un objet à chaque fois que celui-ci sera déplacé. C'est une modification de l'état du problème. KIF étant (entre autres) un langage fonctionnel, on peut voir cette modification comme la production d'un nouvel état à partir d'un ancien état et d'une fonction de changement d'état. On peut voir que les fonctions de changement d'état *aller-en* et *mettre-objet-en*, qui sont les seules par lesquelles un objet peut être déplacé horizontalement, produisent des états dans lesquels la localisation horizontale est différente de celle de l'état précédent lorsque l'objet est déplacé (il est possible d'en faire une démonstration formelle).

C14. La valeur "porté" pour la localisation verticale d'un objet signifie que cet objet est entre les mains du singe. Il s'agit plus d'une explication sémantique que d'une contrainte. On peut quand même remarquer qu'il existe un invariant sur l'état qui dit que l'objet tenu par le singe (attribut du singe) a une localisation verticale "porté".

C15. Pour saisir ou monter sur un objet, le singe doit être à la même localisation horizontale que celui-ci. Cette précondition est présente dans les définitions des fonctions KIF *saisir-objet* et *monter-sur-objet*.

C16. Le singe doit être sur l'échelle pour attraper un objet au plafond. Cette précondition est présente dans la définition de la fonction KIF *saisir-objet*.

C17. Quand un objet est lâché par le singe, sa position horizontale est celle du singe. Cette contrainte se retrouve dans l'expression du résultat de la fonction KIF *lâcher-objet* (on peut en faire une démonstration formelle).

C17bis. Quand un objet est lâché par le singe, sa position verticale est "sol". Cette contrainte se retrouve dans l'expression du résultat de la fonction KIF *lâcher-objet* (on peut en faire une démonstration formelle).

C18. Le singe peut descendre d'un objet tout en tenant un objet. C'est une contrainte positive. Elle conditionne l'absence de précondition relative à ce que le singe tient dans la définition KIF de la fonction *sauter-sur-sol*.

C19. Le singe ne peut pas monter sur un objet s'il en tient un autre. Cette précondition est présente dans la définition de la fonction KIF *monter-sur-objet*.

C20. Le singe ne peut porter plus d'un objet à la fois. Cette contrainte n'est pas explicitement représentée en KIF mais est dérivable de l'invariant qui dit que l'objet désigné par l'attribut du singe "objet porté par le singe", a pour localisation verticale "porté". Des lors, si deux objets physiques avaient "porté" pour valeur de localisation verticale, un au moins serait différent de celui désigné par "objet porté par le singe", ce qui contredirait l'invariant.

C24. Le système ne vérifiera pas la légalité des configurations. Remarquons que le problème ne se pose que pour la configuration (état) initiale, car si celle-ci est valide, on a prouve que les configurations découlant de l'application des fonctions de changement d'état le sont aussi. La spécification KIF donne toutefois une définition de l'état initial qui le rend valide. La contrainte C24 doit-elle être présente telle quelle dans la spécification formelle, ou bien peut-elle être mise en commentaire et n'intervenir qu'à la conception? Dans le premier cas, il ne nous

semble pas possible de la décrire en KIF. Nous n'avons pas de réponse définitive à la question posée, elle reste ouverte.

C25. Le système ne fournira pas d'aide à l'établissement de la configuration initiale. Même remarque que pour C24.

5.4.2. Représentation des fonctions de changement d'état par rapport à leur définition informelle

Les fonctions de changement d'état sont décrites sous la forme de préconditions et de postconditions dans le texte informel. Ces descriptions doivent correspondre à une définition de fonction KIF, dont l'expression a la forme d'un terme (au sens KIF). Pré- et postconditions correspondant à une expression de type "si... alors", le terme sera dès lors un terme conditionnel (voir la forme générale des fonctions de changement d'état au début de ce chapitre).

La représentation des **préconditions** dans ce terme conditionnel est immédiate : c'est la partie "proposition" du terme conditionnel (voir la définition du terme conditionnel au chapitre 3).

La représentation des **postconditions** est moins directe, puisque la fonction de changement d'état produit un terme résultat, qui est un nouvel état. Pour qu'il y ait adéquation de ce nouvel état aux postconditions qu'il doit vérifier, il doit en fait respecter trois types de propositions :

1. la proposition de consistance (qui garantit la cohérence du nouvel état) ;
2. les postconditions informelles ;
3. une postcondition par défaut : "Tous les éléments de l'état autres que ceux mentionnés dans les postconditions informelles restent inchangés". Cette postcondition touche à ce que l'on appelle dans la littérature le "frame problem" ([DUBOIS93], [GEORGEFF87]).

Notons que les points b et c déterminent la nouvelle valeur de chaque élément du nouvel état sans exception et que si les postconditions informelles ont été correctement définies, la proposition de consistance est automatiquement vérifiée.

5.5. EVALUATION GLOBALE DE KIF

5.5.1. Spécification du problème

Au niveau de la spécification du problème, KIF est un langage puissant. Réserve faite de l'aspect "métaconnaissance" du problème que nous traiterons au prochain chapitre (et il existe en KIF des concepts appropriés à la métaconnaissance, que nous n'utiliserons pas), nous avons pu représenter tous les éléments du problème, soit sous forme d'objets KIF (produits cartésiens), soit au travers de relations KIF exprimant des prédicats à un argument, vrais si l'objet KIF passé en argument représente bien l'objet concerné (par exemple, *est-le-singe ?singe*). Les invariants du problème sont également définis (même si, contrairement à un langage de spécification classique comme Z, il n'existe pas de concept d'invariant en tant que tel en KIF), ainsi que les actions que le singe peut opérer, sous forme de fonctions de changement d'état.

Un avantage de KIF est qu'il est possible de traiter l'aspect "information incomplète". En effet, le concept de définition partielle permet de cerner partiellement un objet (ou une relation, ou une fonction) dont on ne connaît pas toutes les caractéristiques. Par exemple, l'objet-cible du problème n'est réellement connu qu'à la configuration initiale, mais on sait de lui qu'il est un des objets physiques et que son poids est "léger". On peut donc donner une définition partielle de l'objet-cible lors de la spécification statique du problème (celle qui est indépendante de la configuration de départ) et une deuxième lors de la spécification dynamique (la partie de la spécification qui dépend de la configuration initiale). Il est possible de montrer que deux définitions partielles d'un même objet ne se contredisent pas, et qu'elles identifient l'objet de façon unique.

5.5.2. Correction de la spécification

A ce niveau, nous nous poserons deux questions.

Le texte KIF contient-il assez d'information pour permettre d'établir des propriétés?

En clair, peut-on trouver toutes les hypothèses nécessaires à l'établissement de preuves de propriétés de la spécification[38] au sein du texte KIF? A la lumière des deux démonstrations que nous avons faites ci-dessus, la réponse semble "oui", car toutes les hypothèses que nous avons dû poser pour montrer la consistance d'une fonction de changement d'état et l'existence d'une solution étaient présentes telles quelles dans le texte KIF. Quant aux thèses de ces mêmes démonstrations, elles sont exprimables sous forme de propositions KIF et peuvent donc être inclues à un texte de spécification.

Nous verrons toutefois au prochain chapitre que cette réponse doit être quelque peu nuancée.

REMARQUE : outre les preuves de corrections intrinsèques à la spécification formelle, il est aussi possible d'utiliser des éléments de la spécification informelle lorsqu'ils ne sont pas ambigus (certaines contraintes, par exemple) et faire la preuve de ce qu'ils sont bien vérifiés dans la spécification KIF. Prenons par exemple les contraintes C17 et C17bis (quand un objet est lâché par le singe, sa position horizontale est celle du singe et sa position verticale est "sol"). Ces contraintes sont traduisibles en KIF et permettent donc d'exprimer la thèse d'une démonstration de correction (et non plus seulement de consistance) de la fonction *lâcher-objet*.

Le langage KIF peut-il être utilisé comme formalisme pour l'écriture des démonstrations?

Ici par contre, la réponse est "non, pas toujours". S'il est possible d'exprimer les hypothèses en KIF (elles sont présentes dans le texte de la spécification) et les thèses (comme la proposition de consistance décrite plus haut), il n'est pas toujours possible de décrire les étapes intermédiaires d'une démonstration. Dans la preuve de l'existence d'une solution, nous avons du recourir au concept de variable au sens

[38]Rappelons qu'il y a plusieurs types de propriétés prouvables : consistance des fonctions de changement d'état, existence d'une solution, existence d'une solution en un nombre borné d'actions, correction des fonctions de changement d'état, cohérence des définitions partielles d'un même objet,...

traditionnel (informatique) du terme pour représenter des valeurs au sein des états que nous avons décrits. Les variables KIF ne pouvaient être utilisées car elles n'ont qu'une portée locale, alors que deux états différents pouvaient contenir des variables de même nom, ce qui signifiait qu'elles représentaient la même valeur dans chaque état (il n'y a pas de lien entre deux variables KIF de même nom dans deux descriptions différentes d'état).

Le langage KIF n'est donc pas un langage permettant d'exprimer des propriétés sur lui-même.

En résumé, nous dirons que KIF contient tous les éléments permettant l'établissement d'une preuve de correction, mais qu'il n'offre pas les outils de réalisation de cette preuve.

5.5.3. Divers

Lisibilité et facilité d'écriture du langage KIF

Le langage KIF n'est sûrement pas facile à lire ni à écrire. C'est un langage dont les principales structures sont celles de LISP, et dans lequel les parenthèses ont une place prépondérante. Une définition de fonction un peu complexe peut voir l'ouverture de plus de 10 niveaux de parenthèses. Il est donc parfois difficile de comprendre la structure réelle d'une phrase KIF lorsqu'on doit compter les parenthèses pour savoir où se trouve la parenthèse fermante correspondant à telle parenthèse ouvrante, par exemple. De plus, lorsqu'on écrit un texte KIF, il n'est pas rare de faire des fautes quant au nombre de parenthèses fermantes et d'en omettre une, ou d'en mettre une de trop (le manuel de référence lui-même est truffé de ces erreurs de parenthésage). Le développement d'un éditeur structurel pour KIF lèverait cette remarque concernant l'écriture de KIF, mais pas celle qui concerne sa difficulté de lecture.

Un autre aspect un peu pénible de la lecture et de l'écriture en KIF est la notation préfixée de toutes les propositions logiques, à laquelle l'être humain est peu habitué. Une telle notation est certes plus rigoureuse que la notation infixée classique, mais elle est plus difficile à déchiffrer. Ainsi, lorsqu'on écrit en KIF *(=> (r1 @args1) (r2 @args2) (r3 @args3))*, cela signifie "(r1 ET r2) => r3" et non "r1 => (r2 ET r3)". Si l'on n'est pas familiarisé avec la notation KIF, le sens de cette proposition prête à confusion.

Il serait intéressant de développer une syntaxe parallèle de KIF, dans laquelle le nombre de parenthèses diminue et qui utilise une notation infixée pour les propositions. Cette syntaxe parallèle est, pensons-nous, axiomatisable en KIF.

Typage des données

En KIF, il n'y a pas de type de données. C'est probablement lié au fait qu'il n'y a pas de variable au sens traditionnel des langages de programmation. Dans le problème du singe et des bananes, cette absence de typage ne nous a pas gêné outre mesure, puisque le nombre de valeurs possibles des éléments qui composent un état du problème est limité (seulement quatre noms d'objets, deux valeurs de poids, etc.). Dans un problème différent, où l'on aurait par exemple besoin d'une chaîne de caractères pour représenter une information, cette absence de type ferait défaut : une énumération de toutes les valeurs possibles serait fastidieuse, et une description sous forme de liste rendrait le texte KIF trop lourd. On pourrait pourtant

développer une axiomatisation formelle des types de données et des fonctions associées à ces types (le manuel de référence 3.0. de KIF [GENESERETH92] précise d'ailleurs que, en ce qui concerne les types numériques, cette axiomatisation est en cours et sera intégrée aux prochaines versions, et propose déjà une série de fonctions numériques). Un type de données pourrait être représenté par une relation KIF, comme c'est déjà le cas pour les types numériques (*integer ?x*, *real-number ?x*).

Ontolingua

Les chercheurs de l'Université de Stanford qui ont développé KIF ne se sont pas arrêtés là, et ont notamment développé un outil permettant de décrire des ontologies, *Ontolingua*. Une ontologie est un ensemble de définitions de primitives de représentation des connaissances : classes, relations, fonctions, objets. Le texte KIF du problème du singe et des bananes est une ontologie. Ontolingua est un mécanisme permettant d'écrire des ontologies sous forme canonique, de façon à pouvoir être facilement traduits dans différents langages et systèmes de représentation. En ce sens, les motivations d'Ontolingua sont similaires à celles de KIF. La syntaxe et la sémantique d'Ontolingua sont d'ailleurs basées sur celles de KIF. Ontolingua permet une représentation plus immédiate des notions de classe, généricité, héritage, etc. Certaines critiques adressables à KIF sont probablement levées pour Ontolingua (lisibilité, représentation difficile de concepts, etc.). Ontolingua est en quelque sorte la version "orientée-objet" de KIF, et il est probable que la transformation d'ALBERT en Ontolingua est plus immédiate que d'ALBERT en KIF.

5.5.4. Conclusion provisoire et perspective

Bien qu'au départ il n'ait pas été conçu dans ce but, KIF nous est apparu comme un langage intéressant pour spécifier un problème typique de représentation des connaissances, le problème du singe et des bananes, qui est un problème de planification.

Il n'offre bien sûr pas tous les concepts présents dans les langages de spécification classique comme Z, mais la plupart d'entre eux sont aisément retrouvables, comme par exemple le concept d'invariant, ainsi que nous l'avons vu. Par ailleurs, on peut envisager la création d'un "KIF étendu" dans lequel ces concepts seraient définissables.

KIF est plus pauvre sur le plan de la démonstration des propriétés d'une spécification, mais rien ne nous empêche d'utiliser un formalisme mathématique classique.

En conclusion, la plupart des concepts de KIF nous paraissent pouvoir jouer un rôle convenable dans la représentation des problèmes de connaissances

Il faut toutefois tenir compte du problème de la représentation de la métaconnaissance (stratégie), que nous avons éludé. Un premier examen du problème et des outils KIF permettant la représentation de la métaconnaissance nous a donné l'impression que la spécification de cet aspect du problème n'est pas possible en KIF. La raison en est que l'explicitation d'une stratégie de résolution est essentiellement de type algorithmique, et que KIF est par contre fonctionnel et déclaratif. C'est pourquoi nous avons décidé de consacrer un chapitre à part à une solution personnelle au problème de la métaconnaissance (c'est le prochain chapitre).

D'autre part, il serait intéressant de spécifier un tout autre type de problème de représentation des connaissances que celui du singe et des bananes. Nous pensons à un problème de diagnostic, par exemple, qui ferait intervenir les concepts de règle d'inférence et de logique non-monotonique, présents en KIF.

Un dernier point pertinent serait de développer une spécification du problème du singe et des bananes dans un langage de spécification différent, comme Z, et voir s'il est possible de prouver la correction de la spécification KIF par rapport à la spécification Z, en supposant celle-ci correcte[39].

Pour terminer, nous rappellerons que cette évaluation garde un caractère provisoire, parce que KIF et les outils greffés autour du langage sont toujours en développement, et qu'il est essentiel de tenir compte de ces développements dans le futur.

---

[39]Le texte produit en ALBERT au chapitre 2 pourrait jouer un rôle dans ce sens, mais rppelons qu'ALBERT ne permet de représenter que les agents du problème et les actions qui sont liées à ces agents.

CHAPITRE 6
PROPOSITION DE REPRESENTATION DE LA MÉTACONNAISSANCE EN KIF

6.1. INTRODUCTION

Dans la spécification KIF du problème du singe et des bananes, ainsi que dans l'évaluation de KIF basée sur cette spécification, nous avons délibérément omis de représenter la métaconnaissance du problème, c'est-à-dire la stratégie à appliquer. Nous allons consacrer ce chapitre au problème de la stratégie dans la recherche d'une solution au problème du singe et des bananes.

Contrairement au reste de la spécification KIF présentée au chapitre 4, le texte que nous allons produire ici ne repose pas directement sur la spécification informelle du problème, mais sur une solution propre respectant dans les grandes lignes le principe de stratégie but/sous-buts (ou "dirigée par le but", suivant le terme de [BUSSENOT92]), sans toutefois l'appliquer à la lettre. On verra notamment que dans cette solution, les notions de but et de sous-but ne sont qu'implicites, et que les outils de base de représentation de la métaconnaissance en KIF ne sont pas utilisés. Par contre, cette solution permettra d'établir un lien entre la notion de "séquence-solution" spécifiée au chapitre 4 et celle de construction d'une solution permettant au singe d'attraper l'objet-cible. Le choix de cette approche se justifie par le caractère évolutif du projet KSL, projet dans lequel les approches "alternatives" du problème de la spécification abstraite d'un système expert restent bienvenues.

Cette vision du problème nous a poussés à séparer complètement la partie "métaconnaissance" du reste de la spécification. La raison en est que, s'agissant d'une solution personnelle, nous estimons ne pas pouvoir donner une évaluation réelle de la représentation de la métaconnaissance en KIF, puisque d'une part nous n'utilisons pas les outils standards que KIF offre à ce sujet[40], et que d'autre part nous nous éloignons de la spécification informelle. Cela ne veut pas dire que nous nous abstiendrons d'évaluer la représentation de la métaconnaissance que nous allons effectuer ici, mais que cette évaluation sera moins celle de KIF en tant que langage de spécification, comme précédemment, que celle de la solution elle-même. Cela veut également dire que cette évaluation ne prétendra pas cerner définitivement le problème de la métaconnaissance dans une spécification abstraite en KIF, puisqu'on peut imaginer d'autres solutions permettant de spécifier correctement la stratégie à appliquer au problème du singe en KIF.

Nous commencerons par rappeler le concept de stratégie "orientée-buts" d'un problème de planification (en l'illustrant par le problème du singe), puis nous proposerons notre solution de représentation de la stratégie du problème en montrant comment elle se rattache au concept "orienté-buts". Ensuite, nous

[40]Pour plus de détails sur ces outils, voir la section 3.4. et [GENESERETH92], chapitre 9.

montrerons que la séquence d'actions construite en appliquant cette stratégie est bien une "séquence-solution" au sens défini dans la spécification du chapitre 4. Enfin, nous évaluerons la solution proposée et sa représentation en KIF.

6.2. LA STRATEGIE "ORIENTEE-BUTS"

REMARQUE PRELIMINAIRE : Le but de cette section est d'introduire la notion de stratégie "orientée-buts" afin que le lecteur soit familiarisé avec les principes de ce concept. Nous avons cependant déjà dit que la solution que nous avons retenue pour représenter cette stratégie en KIF n'en conserve que l'idée générale et s'en éloigne dans les détails, notamment en ce qui concerne la notion de sous-but qui n'y apparaît que de façon implicite. Dans la prochaine section, où nous décrirons notre proposition de représentation de la métaconnaissance du problème du singe en KIF, nous mettrons l'accent sur les rapprochements entre cette stratégie et la façon dont nous l'appliquerons.

La stratégie "orientée-buts" est celle qu'on applique généralement dans les problèmes de planification. Son principe est qu'il existe un ***but*** à satisfaire, et qu'il faut déterminer l'action finale à accomplir pour que ce but soit atteint. Pour pouvoir être exécutée, cette action a des préconditions qui doivent être satisfaites. Ces préconditions deviennent les ***sous-buts*** à satisfaire. On choisit[41] alors une action (et une liste de paramètres associés) qui permet de satisfaire un ou plusieurs de ces sous-buts, c'est-à-dire telle que l'état résultant de son application vérifie les sous-buts en question. Ceux-ci sont alors **résolus**. L'action choisie génère une nouvelle série de sous-buts à satisfaire (correspondant à ses propres préconditions), que l'on ajoute aux sous-buts non encore résolus. Le processus choix-action/génération-de-sous-buts est réitéré jusqu'à ce que l'état de départ du problème satisfasse tous les sous-buts en cours. La solution du problème est la liste d'actions ainsi choisies, dans l'ordre inverse de celui de sa production.

Prenons l'exemple du singe et des bananes et supposons une configuration de départ dans laquelle :

- le singe est en 5,5 sur le sol et ne porte rien
- l'échelle est en 8,4 sur le sol et aucun objet n'est posé sur elle
- les bananes sont au plafond en 7,1

Le but du problème est que le singe tienne les bananes. L'action finale à accomplir est que le singe prenne les bananes, c'est-à-dire *saisir-objet (bananes)*. Pour pouvoir accomplir cette action, il faut que les préconditions en soient satisfaites. Ces préconditions[42], qui deviennent les nouveaux sous-buts du problème, sont :

1. le singe est en 7,1 (même position horizontale que les bananes)
2. le singe est sur l'échelle (puisque les bananes sont au plafond)

---

[41]Le choix du sous-but à satisfaire peut poser problème. Nous envisagerons cet aspect dans la spécification en KIF.

[42]Nous ne prenons en compte que les préconditions significatives. Les autres préconditions (par exemple, que l'objet que le singe veut prendre est bien un objet physique) sont vérifiées par le choix de l'action à opérer et de ses paramètres.

3. le singe ne porte rien (puisqu'il ne peut prendre plus d'un objet à la fois)
4. il n'y a rien de posé sur les bananes

Une action applicable pour que l'un de ces sous-buts (le 2) soit satisfait est l'action *monter-sur (échelle)*. Les préconditions à satisfaire sont :

5. le singe est sur le sol
6. l'échelle est sur le sol
7. le singe et l'échelle sont à la même position horizontale
8. le singe ne porte rien

Il faut rajouter à cette liste les sous-buts non encore satisfaits, c'est-à-dire (outre que le singe ne porte rien, déjà présent ci-dessus) :

9. le singe est en 7,1
10. il n'y a rien sur les bananes

L'action qui permettrait de satisfaire les sous-buts 5, 6, 7, 8 et 9 est l'action *mettre-objet-en (échelle, (7,1))* dont les préconditions à satisfaire sont :

11. le singe est sur le sol
12. le singe tient l'échelle

auxquelles il faut rajouter le sous-but non encore résolu

13. il n'y a rien sur les bananes

Les sous-buts 11 et 12 peuvent être résolus par l'action *saisir-objet (échelle)* dont les préconditions, qui deviennent les nouveaux sous-buts à résoudre, sont :

14. le singe est sur le sol et (14 bis) l'échelle n'est pas au plafond
15. le singe est en (8,4), à la même position horizontale que l'échelle
16. le singe ne tient rien
17. il n'y a rien sur l'échelle

auxquels on rajoute le sous-but non résolu

18. il n'y a rien sur les bananes

Les sous-buts 14 et 15 peuvent être résolus par l'action *aller-en (8,4)*. La précondition en est :

19. le singe est sur le sol

à laquelle on rajoute les sous-buts non résolus :

20. le singe ne tient rien
21. il n'y a rien sur l'échelle
22. l'échelle n'est pas au plafond
23. il n'y a rien sur les bananes

L'ensemble des sous-buts 19 à 23 est résolu, c'est-à-dire qu'ils sont tous vérifiés dans l'état initial du problème. La solution du problème consiste donc en la séquence d'actions que nous venons de décrire en commençant par la dernière. Si on reprend le raisonnement en sens inverse, on voit en effet que:

- le singe peut **aller en (8,4)**
- il peut dès lors **saisir l'échelle**

- il peut ensuite **mettre l'échelle en (7,1)**
- il peut ensuite **monter sur l'échelle**
- il peut enfin **saisir les bananes** et son but est atteint.

Le lecteur familiarisé avec les concepts de base de l'Intelligence Artificielle aura compris que cette stratégie n'est rien d'autre qu'une application de la méthode dite de "chaînage arrière" ([JACKSON90]).

Remarquons d'ailleurs que cette méthode de construction est celle que l'être humain applique plus ou moins consciemment lorsqu'il vise un but et qu'il doit produire une série d'actions pour atteindre ce but.

6.3. REPRESENTATION DE LA STRATEGIE EN KIF

6.3.1. Idée générale

La solution du problème du singe est, comme on l'a vu au chapitre 4, représentée par une *séquence d'actions* et de paramètres, chaque action correspondant à une des fonctions de changement d'état décrites. L'*exécution* de cette séquence à partir d'un état initial donné (c'est-à-dire l'application de la première action à cet état initial, puis l'application de la deuxième action à l'état produit par la première action, etc.) produit un *état-but*, c'est-à-dire un état du problème dans lequel le but (le singe tient l'objet-cible) est atteint.

Construire cette solution suivant la stratégie "orientée-buts" revient à déterminer une par une les actions à appliquer en partant de la dernière, c'est-à-dire celle qui, partant d'un *état-non-but* (un état dans lequel le but n'est pas atteint), produit un état-but.

Examinons cette dernière action (ou fonction). L'état final du problème, qu'elle génère, doit être un état-but, donc appartenir à l'ensemble des états-buts. Son état de départ doit quant à lui être un état-non-but (sinon, le but serait déjà atteint à l'étape précédente). Puisqu'il existe un *ensemble* d'états-buts, il y a nécessairement un *ensemble* d'états-non-buts tel que l'application de la fonction à un des états de cet ensemble génère un état de l'ensemble des états-buts.

Prenons l'exemple du problème du singe où les bananes, qui sont l'objet-cible, se trouvent au plafond en (7,1). L'ensemble des états-buts contient tous les états du problème dans lesquels le but est réalisé, donc tous les états dans lesquels le singe tient les bananes. Il existe bien une fonction qui permet de passer d'un état-non-but (dans lequel le singe ne tient pas les bananes) à un état-but : c'est la fonction *saisir-objet (bananes)*. Il est d'ailleurs évident que cette fonction correspond à la dernière action que le singe devra entreprendre pour atteindre son but. L'ensemble des états-non-buts auxquels on peut appliquer la fonction *saisir-objet (bananes)*[43] est celui de tous les états qui respectent les

[43]Par abus, nous parlerons de fonctions sur des ensembles d'états et produisant des ensembles d'états. Il est évident qu'une fonction de changement d'état en soi s'applique à un seul état et produit un seul état, mais on simplifiera le texte en parlant d'ensembles d'états auxquels on peut appliquer la fonction, et d'ensemble d'états produits. L'application de la fonction à chaque état de l'ensemble de départ produit un état de l'ensemble d'arrivée.

préconditions de cette fonction, afin qu'elle soit applicable. Ces préconditions sont les suivantes :

- le singe est en 7,1
- le singe est sur l'échelle
- le singe ne porte rien
- il n'y a rien de posé sur les bananes

Ces préconditions correspondent aux sous-buts générés par la première étape de la résolution du problème, comme nous l'avons montré à la section précédente.

Ayant déterminé la dernière action à accomplir, on peut recommencer le procédé à partir d'un nouvel ensemble d'états-buts qui contient tous les états vérifiant les sous-buts. Il faut alors redéterminer une action applicable ainsi que le nouvel ensemble d'états-non-buts correspondant, ensemble qui devient lui même le nouvel ensemble d'états-buts, et ainsi de suite. La solution est entièrement construite lorsque l'état de départ appartient à l'ensemble des états-non-buts

En pratique, la solution KIF que nous allons décrire ne parlera pas de sous-buts. Elle ne se présentera qu'en termes d'*ensembles d'états* et de fonctions à appliquer pour passer d'un ensemble à l'autre. Le lien avec les buts et sous-buts est cependant immédiat puisque ces ensembles d'états contiendront toujours des états vérifiant les sous-buts en cours.

Enfin, la solution KIF devra tenir compte d'un problème de choix, lorsque plusieurs actions sont applicables pour générer le même ensemble d'états-buts à partir d'ensembles d'états-non-buts différents.

6.3.2. Spécification KIF

Fonctions génératrices d'un ensemble d'états

Afin de déterminer l'action à accomplir pour générer un état-but, il faut connaître l'ensemble des fonctions qui permettent de générer un ensemble d'états-buts à partir d'un ensemble d'états-non-buts. C'est parmi celles-ci que l'on devra choisir l'action à appliquer.

Voici la définition de la fonction KIF qui produit l'ensemble des fonctions de changement d'état permettant de générer un ensemble d'états donnés :

```
(DEFFUNCTION fonctions-génératrices (?ensemble-d-états) :=
   (SETOFALL (?f)
             (AND (est-une-fonction-du-problème ?f)
                  (EXISTS (?état @args)
                          (AND (NOT (member ?état ?ensemble-d-états))
                               (member (apply ?f (LISTOF ?état @args))
                                       (?ensemble-d-états)))))))
```

En français : les fonctions génératrices d'un ensemble d'états donnés sont l'ensemble des fonctions telles que :

- chacune d'entre elles est une fonction du problème
- pour chacune d'entre elles, il existe un état et une suite d'arguments tels que :

- cet état n'appartient pas à l'ensemble d'états donnés
- l'application de la fonction à cet état et à ces arguments est un état élément de l'ensemble d'états donnés.

Grâce à cette fonction KIF, nous pouvons donc connaître l'ensemble des fonctions de changement d'état qui permettent de générer un ensemble d'états-buts. Il reste à déterminer celle que l'on va réellement appliquer, et c'est ici que se pose le problème du choix.

Le problème du choix

A partir d'un ensemble de fonctions qui, à partir de différents ensembles d'états-non-buts, permettent de générer un ensemble d'états-buts, il faut en choisir une que nous rajouterons (par la fin, pour rappel) à la liste d'actions qui constitue la solution du problème. Prenons en effet pour ensemble d'états-buts l'ensemble des états dans lesquels le singe est sur l'échelle et ne porte rien. D'une part, la fonction *monter-sur-objet (échelle)* permet bien de générer cet ensemble d'états-buts à partie d'un ensemble d'états-non-buts dans lesquels le singe est au pied de l'échelle et ne porte rien. D'autre part, la fonction *lâcher-objet (objet-porté-par-le-singe)* permet aussi de générer cet ensemble d'états-buts, avec cette fois ci pour ensemble d'états de départ celui de tous les états dans lesquels le singe est déjà sur l'échelle et porte un objet.

Laquelle de ces fonctions choisir ? D'une part, il faut tenir compte de l'aspect heuristique de cette question de choix. D'autre part, il faut envisager la possibilité qu'une action choisie, en produisant un nouvel ensemble d'états-buts inatteignable à partir de l'état de départ, ne permette pas d'obtenir une solution au problème et qu'il faille donc envisager un "retour en arrière" en choisissant une autre fonction que celle d'abord envisagée.

Face à ce double aspect du choix de la fonction, KIF n'offre pas de réponse. D'une part, la définition de règles heuristiques ("il vaut mieux choisir telle fonction plutôt que telle autre" ou "appliquer telle fonction le plus tôt possible") n'existe pas en KIF. D'autre part, KIF, qui est résolument fonctionnel, est par conséquent déterministe. Toute fonction KIF qui choisirait une fonction de changement d'état ne permettrait pas un "deuxième choix" si le premier débouchait dans une impasse.

La solution que nous avons retenue est la suivante : nous définissons une fonction KIF *meilleure-fonction* qui permet de choisir la meilleure fonction parmi un choix de fonctions génératrices d'un ensemble d'états-buts. Afin d'éviter l'aspect déterministe de ce choix, la définition de *meilleure-fonction* ne sera que partielle, et contiendra le minimum d'informations à son sujet, c'est-à-dire simplement que son résultat est une fonction parmi l'ensemble de fonctions qu'elle reçoit comme arguments. Quant aux heuristiques de choix, impossibles à décrire en KIF, nous les rajouterons en commentaire sous cette définition[44].

Outre l'ensemble de fonctions parmi lesquelles choisir, un autre argument nous semble devoir intervenir dans la description de *meilleure-fonction* : l'ensemble des états-buts à générer, qui intervient aussi dans le choix d'une

[44]Comme pour les invariants du problème (chapitre 4), nous ne donnerons pas une liste exhaustive d'heuristiques, mais seulement un exemple. L'ingénieur chargé de la spécification réelle du problème devra être plus complet.

fonction (autrement, la meilleure fonction ne dépendrait que de l'ensemble des fonctions, ce qui reviendrait à supposer qu'il y a un ordre de priorité absolue sur les fonctions de changement d'état ; il nous semble au contraire que le choix d'une fonction par rapport à une autre dépend du contexte, donc des états-buts que ces fonctions génèrent).

Voici la définition KIF de *meilleure-fonction*, suivi d'exemples d'heuristiques (l'heuristique relative au problème du bouclage est fondamentale, comme on le verra dans la section 6.4.) :

```
(DEFFUNCTION meilleure-fonction (?ensemble-de-fonctions ?ensemble-d-états)
    (member (meilleure-fonction (?ensemble-de-fonctions ?ensemble-d-états))
            ?ensemble-de-fonctions))

* HEURISTIQUE *
* Si le singe tient un objet, lui faire lâcher le plus vite possible, donc choisir la *
* fonction (lâcher-objet) le plus tard possible (puisque la solution se construit en *
* commençant par la fin).*

*HEURISTIQUE IMPORTANTE *
* Il faut éviter de choisir une fonction qui produise un ensemble d'états de départ déjà généré
précédemment, ou un sous-ensemble de celui-ci, pour éviter le bouclage. *
```

Arguments générateurs

Après avoir choisi la fonction à appliquer, il faut encore déterminer les arguments de cette fonction. On peut supposer que, pour un même ensemble d'états-buts, une fonction puisse être appliquée à plusieurs listes d'arguments différentes, et un problème de choix se pose à nouveau[45]. Il faut donc d'abord déterminer l'ensemble des listes d'arguments applicables par une même fonction pour générer un même ensemble d'états-buts, ensuite effectuer un choix parmi ces listes.

Voici la définition de la fonction KIF qui produit l'ensemble des listes d'arguments qui, appliqués par une fonction donnée, permettent de générer un ensemble d'états donnés :

```
(DEFFUNCTION arguments-générateurs (?f ?ensemble-d-états) :=
    (SETOFALL (LISTOF @args)
              (EXISTS (?état)
                      (member (apply ?f (LISTOF ?état (LISTOF ?@args))
                                     ?ensemble-d-états))))
```

En français : l'ensemble des arguments générateurs d'un ensemble d'états donnés pour une fonction donnée est l'ensemble de toutes les listes d'arguments telles que pour chacune d'entre elles, il existe un état du problème tel qu'en appliquant la fonction donnée à la liste composée de cet état et de la liste d'arguments, on obtienne un état appartenant à l'ensemble des états donnés.

Le problème du choix des arguments pour une fonction donnée est similaire à celui du choix de la fonction elle-même. Nous appliquerons donc la même

---

[45]Pratiquement, dans le problème du singe, il nous semble que le problème ne se rencontre pas (une seule liste d'arguments par fonction pour générer un même ensemble d'états-buts), mais nous envisagerons la possibilité de devoir choisir une liste d'arguments, d'une part parce que nous ne pouvons pas prouver l'unicité de cette liste d'arguments, d'autre part pour être plus général.

solution, qui est de donner une définition partielle minimale de ces arguments et une série d'heuristiques en commentaire (laissé à l'appréciation de l'expert). Voici cette définition KIF :

```
(DEFFUNCTION meilleure-liste-d-args (?f ?ensemble-de-listes-d'args ?ensemble-d-états)
    (member meilleure-liste-d-args ?ensemble-de-listes-d'args)

* HEURISTIQUES *
* <au choix de l'expert> *
```

Pour illustrer ce problème de choix, imaginons une variante du problème du singe dans laquelle, pour prendre un objet, il suffit au singe d'être à une position proche de cet objet (soit, si l'objet est en [5,5], les positions [4,4], [4,5], [4,6], [5,4], [5,5], [5,6], [6,4], [6,5] et [6,6]). Si le but est que le singe prenne l'objet, un sous-but sera qu'il se trouve à une des positions proches de celui-ci, et pour résoudre ce problème, on pourra appliquer l'action *aller-en* à n'importe laquelle de ces positions. Le choix de la position (argument de *aller-en*) sera fonction d'heuristiques comme "préférer la position géographiquement la plus proche de celle du singe" ou "préférer la position même de l'objet sauf s'il s'agit du lit", etc.

Dernière action d'une séquence

Nous pouvons maintenant définir la *dernière action d'une séquence-solution*. Rappelons que cette action a la forme d'une liste composée d'une fonction de changement d'état et d'une sous-liste d'arguments applicables par cette fonction. La fonction de changement d'état est celle que nous aurons choisie parmi les fonctions génératrices de l'ensemble des états-buts, et la liste d'arguments est celle que nous aurons choisie parmi les arguments générateurs de cet ensemble.

Voici la définition KIF de *dernière-action* :

```
(DEFFUNCTION dernière-action (?ensemble-d-états) :=
    (IF (AND (EXISTS (?f)
                  (= ?f (meilleure-fonction (fonctions-génératrices ?ensemble-d-états)
                                            ?ensemble-d-états)))
             (EXISTS (LISTOF @args)
                  (= LISTOF @args
                     (meilleure-liste-d-args ?f
                                             (arguments-générateurs ?f
                                                                    ?ensemble-d-états)
                                             ?ensemble-d-états))))
        (LISTOF ?f (LISTOF @args))))
```

En français : si, pour un ensemble d'états donnés, il existe une fonction qui soit la meilleure des fonctions génératrices de cet ensemble d'états, et une liste d'arguments qui soit la meilleure des listes d'arguments générateurs de cet ensemble d'états pour cette fonction, alors la dernière action à accomplir pour générer cet ensemble d'états est décrite par une liste composée de cette fonction et de cette liste d'arguments.

En reprenant l'exemple du singe, lorsque le but est de prendre les bananes (et donc que l'ensemble des états-buts contient tous les états dans lesquels le singe tient les bananes), on a bien :

- une fonction génératrice de cet ensemble d'états : *saisir-objet* ;

- une liste d'arguments applicables : *[bananes]* (liste d'un seul item).

Dès lors, la fonction *dernière action* appliquée à l'ensemble des états dans lesquels le singe tient les bananes produit la liste *[saisir-objet [bananes]]*, qui est conforme à la description d'une action donnée au chapitre 4.

Etats de départ d'une action

Puisque la construction d'une solution se fait à partir d'un ensemble final d'états-buts, puis en remontant progressivement vers un ensemble d'états auquel appartient l'état initial du problème, et puisque cette construction se fait action par action, en prenant chaque fois pour ensemble d'états-buts l'ensemble d'états de départs de l'action précédente dans l'ordre de construction, il faut pouvoir déterminer, à partir d'une action et de l'ensemble d'états qu'elle produit, l'ensemble d'états de départ de cette action, c'est-à-dire ceux auquel on peut appliquer l'action pour produire un état de l'ensemble des états finals donnés (et connus) de l'action. Voici la définition KIF de la fonction qui produit cet ensemble d'états de départ sur base d'une action et d'un ensemble d'états finals donnés :

```
(DEFFUNCTION ensemble-des-états-de-départ (?action ?ensemble-des-états-finals) :=
    (IF (EXISTS (?f @args)
                (AND (est-une-fonction-du-problème ?f)
                     (= (LISTOF ?f (LISTOF @args)) ?action)))
        (SETOFALL (?état)
                  (member (apply ?f (LISTOF ?état @args)) ?ensemble-des-états-finals))))
```

En français : l'ensemble des états de départ d'une action donnée qui produit un ensemble d'états finals donnés à partir de ces états de départ s'obtient comme suit : s'il existe une fonction et une liste d'arguments tels que cette fonction est bien une fonction du problème et que la liste composée de cette fonction et de la liste d'arguments est bien l'action donnée, alors c'est l'ensemble des états de départ tels que l'état résultant de l'application de la fonction à l'un de ces états de départ appartient quant à lui à l'ensemble des états finals donnés.

Dans l'exemple où les bananes sont au plafond, en (7,1) et où l'action à exécuter est *[saisir-objet [bananes]]*, il existe bien une fonction (*saisir-objet*) et une liste d'arguments (*[bananes]*) tels que cette fonction est une fonction du problème, que la liste composée de cette fonction et de cette liste d'arguments est bien l'action donnée. Dès lors, l'ensemble des états de départ de cette action est celui de tous les états auxquels on peut appliquer *[saisir-objet [bananes]]*. Ces états sont ceux dans lesquels les préconditions de la fonction *saisir-objet (bananes)* sont respectés.

Solution du problème

Nous disposons maintenant de tous les outils pour construire la liste d'actions qui est la solution du problème, c'est-à-dire celle qui permettra au singe de prendre l'objet-cible.

Comme il s'agit d'une liste d'actions, nous la construirons de façon récursive en déterminant la dernière action pour générer l'ensemble des états-buts du problème et en recommençant la construction pour générer cette fois l'ensemble des états de départ de cette dernière action. L'arrêt de la construction se fait lorsque l'état initial est l'un des états de départ générés.

Voici la définition KIF de *construire-solution* :

```
(DEFFUNCTION construire-solution (?état-initial ?ensemble-d-états-buts) :=
    (IF (member ?état-initial ?ensemble-d-états-buts)
        (LISTOF)
        (append (construire-solution ?état-initial
                                     (ensemble des états de départ
                                         (dernière-action ?ensemble-d-états-buts)
                                         ?ensemble-d-états-buts))
                (LISTOF (dernière-action ?ensemble-d-états-buts)))))
```

En français : la construction de la liste d'actions solution du problème à partir d'un état initial et d'un ensemble d'états-buts se déroule de façon récursive de la manière suivante :

- si l'état initial appartient déjà à l'ensemble des états-buts, la liste est vide
- sinon, ajouter la dernière action de la solution à la construction d'une solution pour, à partir du même état initial, générer l'ensemble des états de départ de cette dernière action.

Définition de la solution optimale du problème

Dans le chapitre 4, nous avions défini une relation "est solution du problème". Toute séquence d'actions permettant au singe, à partir de l'état initial, d'atteindre son but (prendre l'objet-cible) vérifie cette relation.

Nous venons de donner la stratégie à appliquer pour construire une solution optimale, c'est-à-dire orientée-buts. Cette stratégie produit une séquence d'actions donnée, qui est supposée être la meilleure solution du problème.

Nous pouvons donc définir un objet "solution optimale du problème" comme étant la séquence d'actions construite de cette façon. Voici le texte KIF de cette définition :

```
(DEFOBJECT solution-optimale-du-problème :=
    (construire-solution état-initial-du-problème ensemble-des-états-buts))
```

En français : la solution optimale du problème est celle obtenue en appliquant la fonction *construire-solution* à l'état initial du problème et à l'ensemble des états-buts (rappelons que l'un et l'autre sont des objets préalablement définis).

Il est évident que cette solution optimale doit vérifier la relation *est-solution-du-problème*, ce que nous exprimerons par la proposition suivante :

```
(est-solution-du-problème solution-optimale-du-problème)
```

Nous démontrerons cette proposition dans la section suivante.

Exemple

Si nous reprenons l'exemple de configuration initiale de la section 6.2., c'est-à-dire un état initial où :

- le singe est en 5,5 sur le sol et ne porte rien ;
- l'échelle est en 8,4 sur le sol et aucun objet n'est posé sur elle ;
- les bananes (objet-cible) sont au plafond en 7,1 ;
- les caractéristiques des autres objets sont bien sûr définies, mais sans intérêt pour nous.

on peut appliquer la fonction *construire-solution* à cet état initial et à l'ensemble des états-buts défini au chapitre 4 comme étant celui de tous les états dans lesquels le singe tient les bananes en main, et montrer que, à supposer les choix successifs de fonctions et d'arguments corrects par rapport aux heuristiques de choix, elle produit comme résultat la séquence d'actions suivante[46] :

- *[aller-en [8,4]]*
- *[saisir-objet ['échelle]]*
- *[mettre-objet-en ['échelle,7,1]]*
- *[monter-sur ['échelle]]*
- *[saisir-objet ['bananes]]*

séquence qui correspond à la solution construite à la section 6.2.

6.4. PREUVE DE CORRECTION DE LA SOLUTION

Dans cette section, nous allons montrer que la séquence d'actions produite comme solution du problème par la stratégie définie ci-dessus est bien une solution du problème au sens spécifié dans le chapitre 4. Le but de cette section est double : d'une part, elle permettra de confirmer l'évaluation de KIF sur le plan "correction de la spécification" (5.5.2.) par une nouvelle démonstration qui n'utilisera que des éléments de la spécification KIF comme hypothèses et thèses. A ce titre, cette démonstration remplira le même rôle que les deux démonstrations du chapitre 5; D'autre part, elle permettra de valider la représentation de la stratégie que nous avons proposée, au moins sur le plan formel (correction de l'apport "métaconnaissance" par rapport au reste de la spécification KIF).

6.4.1. Thèse

Comme annoncé à la section précédente, la thèse globale de cette démonstration est exprimée par la proposition KIF suivante :

(est-solution-du-problème solution-optimale-du-problème)

qui signifie que la solution optimale du problème, générée par la fonction *construire-solution*, vérifie bien la relation *est-solution-du-problème*, c'est-à-dire qu'elle est bien une séquence d'actions dont l'exécution, à partir de l'état initial du problème, produit un état dans lequel le singe tient l'objet-cible.

Pour les besoins de la démonstration, nous allons définir une relation KIF plus générale de solution qui indiquera qu'une séquence d'action donnée permet, à partir d'un état de départ donné, de produire un état élément d'un ensemble

[46]Nous n'entrerons pas dans les détails (assez fastidieux) de la construction. Le lecteur désireux de vérifier par lui-même l'adéquation de notre solution par rapport à la stratégie appliquée dans l'exemple décrit à la section 6.2. pourra appliquer la construction de la solution en détail, à titre d'exercice.

d'états donné (et non plus obligatoirement l'ensemble des états-buts du problème). Voici cette définition[47] :

(DEFRELATION est-solution-générale (?seq ?état-de-départ ?ensemble-d-états) :=
(member (exécuter-séquence ?seq ?état-de-départ) ensemble-d-états))

En français : une séquence d'actions est une solution générale du problème à partir d'un état de départ donné et par rapport à un ensemble d'états donnés si l'exécution de cette séquence à partir de l'état de départ produit un état qui appartient à l'ensemble d'états donné.

Montrons que, si cette relation est vérifiée pour la séquence d'actions décrite par *solution-optimale-du-problème* et pour l'état initial du problème et l'ensemble des états-buts définis dans la spécification, alors

(est-solution-du-problème solution-optimale-du-problème)

est vérifié.

Si la proposition

(est-solution-générale solution-optimale-du-problème état-initial-du-problème ensemble-des-états-buts)

est vraie, alors, par définition de *est-solution-générale*, on a la proposition suivante :

(member (exécuter-séquence solution-optimale-du-problème état-initial-du-problème)
ensemble-des-états-buts)

De cette proposition, et par définition de la relation *est-état-solution-du-problème*, on peut déduire la proposition suivante :

(est-état-solution-du-problème (exécuter-séquence solution-optimale-du-problème
état-initial-du-problème))

et de cette proposition, par définition la relation de *est-solution-du-problème*, on peut déduire la proposition

(est-solution-du-problème solution-optimale-du-problème)

ce qui est la thèse de notre démonstration globale. Il reste donc à démontrer la proposition de départ, que nous prendrons comme nouvelle thèse, à savoir :

(est-solution-générale solution-optimale-du-problème état-initial-du-problème ensemble-des-états-buts)

6.4.2. Cas de base

La démonstration se fera par récurrence. Il nous faut donc d'abord démontrer le cas de base qui est celui ou l'état initial du problème est déjà un état-but. Nous avons donc pour hypothèse la proposition suivante :

(member état-initial-du-problème ensemble-des-états-buts)

De cette proposition, nous pouvons déduire, par la définition de la fonction *construire-solution*, que le terme de fonction

(construire-solution état-initial-du-problème ensemble-des-états-buts)

---

[47]Cette définition n'apparaîtra pas dans le texte KIF final, parce qu'elle n'a d'utilité qu'au sein de la démonstration en cours.

qui est, par définition de *solution-optimale-du-problème*, l'objet *solution-optimale-du-problème*, est égal à une liste vide (*LISTOF*). Dès lors, par définition de *exécuter-séquence*, le terme fonctionnel

(exécuter-séquence solution-optimale-du-problème état-initial-du-problème)

est égal à l'état initial du problème puisqu'on exécute une séquence d'actions vide à partir de cet état initial.

On a par hypothèse que l'état initial du problème appartient à l'ensemble des états-buts. En remplaçant cet état initial par le terme fonctionnel ci-dessus, qui lui est égal, on obtient la proposition

(member (exécuter-séquence solution-optimale-du-problème état-initial-du-problème)
ensemble-des-états-buts)

De cette proposition, et par définition de la relation *est-solution-générale*, on peut déduire la proposition

(est-solution-générale solution-optimale-du-problème état-initial-du-problème ensemble-des-états-buts)

C.Q.F.D.

6.4.3. Cas de récurrence

Le problème de la récurrence

Pour pouvoir construire cette démonstration par récurrence, il faut admettre qu'il y ait récurrence, donc que la fonction *construire-solution* élabore une séquence finie d'actions et rencontre tôt ou tard son cas de base. Or, bien que nous ayons montré au chapitre 5 que cette solution en une séquence finie d'actions existe, rien ne dit que *construire-solution* la produit bien, parce que nous n'avons pas assez d'information formelle sur le choix de la fonction à appliquer à chaque étape (comme on l'a vu, ce choix est non-déterministe et fonctions d'heuristiques). Nous devrons nous baser sur l'heuristique de choix qui impose qu'on ne choisisse jamais une action générant comme ensemble d'états de départs un ensemble d'états déjà générés (ou sous-ensemble de celui-ci), afin d'éviter la circularité[48]. Cela induit que tout nouvel ensemble d'états générés contient au moins un état non contenu par son prédécesseur. Comme le nombre d'états total est fini, on générera tôt ou tard un ensemble qui contiendra l'état initial du problème, ce qui est la cas de base de la fonction récursive *construire-solution*.

Démonstration du cas de récurrence

La fonction *construire-solution*, comme on l'a vu, élabore une solution de la manière suivante : elle détermine la dernière action de la séquence qu'elle construit, puis s'appelle de façon récursive pour calculer le reste de la séquence avec l'ensemble des états de départ de la dernière action pour ensemble d'états-buts. Notre hypothèse de récurrence sera donc que ce "reste de la séquence"

---

[48]De ce fait, on peut même dire qu'il ne s'agit pas d'une heuristique, mais bien d'une règle incontournable. Elle reste toutefois difficile (si pas impossible) à exprimer en KIF, et nous la laisserons en commentaire.

vérifie la relation *est-solution-générale* par rapport à son propre ensemble d'états-buts (l'ensemble des états de départ de la dernière action), ce qui s'exprime par la proposition suivante :

```
(est-solution-générale
    (construire-solution état-initial
                (ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)
                                              ?ensemble-d-états)))
    état-initial
    (ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)))
```

quel que soit l'ensemble d'états représenté par *?ensemble-d-états*, à l'exception de ceux qui contiennent l'état initial (qui correspondent au cas de base démontré) . La thèse à en déduire sera

```
(est-solution-générale (construire-solution état-initial-du-problème ?ensemble-d-états)
                       état-initial-du-problème
                       ?ensemble-d-états)
```

puisque dans le cas particulier où *?ensemble-d-états* représente l'ensemble des états-buts du problème, le terme

```
(construire-solution état-initial-du-problème ?ensemble-d-états)
```

est égal à l'objet *solution-optimale-du-problème* par définition de celui-ci, et qu'en remplaçant ce terme par cet objet et en instanciant *?ensemble-d-états*, on obtient la thèse générale, à savoir :

```
(est-solution-générale solution-optimale-du-problème état-initial-du-problème ensemble-des-
états-buts)
```

Passons à cette démonstration. Nous avons donc la proposition suivante pour hypothèse :

```
(est-solution-générale
    (construire-solution état-initial
                (ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)
                                              ?ensemble-d-états)))
    état-initial
    (ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)))
```

En développant la relation *est-solution-générale* suivant sa définition, nous pouvons déduire la proposition suivante :

```
(member
    (exécuter-séquence
        (construire-solution état-initial
                    (ensemble-des-états-de-départ (dernière-action ?ensemble-d-
états)
                                                  ?ensemble-d-états))
        état-initial)
    (ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)))          (a)
```

Cette proposition signifie que si on exécute la séquence construite par la fonction *construire-solution* avec pour arguments l'état initial du problème et un ensemble d'états finals qui sont les états de départ de la dernière action à fournir pour générer *?ensemble-d-états*, alors on génère bien un de ces états de départ.

D'autre part, si on développe

(construire-solution état-initial ?ensemble-d-états)

suivant la définition de *construire-solution*, et puisque l'état initial n'appartient pas à l'ensemble d'états donnés, on obtient le terme suivant :

(append (construire-solution état-initial
(ensemble-des-états-de-départ (dernière-action ?ensemble-d-états)
ensemble-d-états))
(LISTOF (dernière-action ?ensemble-d-états)))))

Ce terme est la concaténation de deux listes. Appliquer la fonction *exécuter-séquence* à cette concaténation de deux listes à partir de l'état initial revient à l'appliquer sur la deuxième liste à partir de l'état produit par l'application d'*exécuter-séquence* à la première liste et à l'état initial[49].Dès lors,

(exécuter-séquence (construire-solution état-initial ?ensemble-d-états)) **(b)**

est égal à

(exécuter-séquence (LISTOF (dernière-action ?ensemble-d-états))
(exécuter-séquence
(construire-solution état-initial
(ensemble-des-états-de-départ
(dernière-action ?ensemble-d-états)
ensemble-d-états))
état-initial)) **(c)**

Or nous savons par **(a)** que l'état produit par

(exécuter-séquence (construire-solution état-initial
(ensemble-des-états-de-départ
(dernière-action ?ensemble-d-états)
ensemble-d-états))
état-initial))

que nous appellerons *?état-1* par commodité, appartient à l'ensemble des états de départ de *(dernière-action ?ensemble-d-états)* . Nous pouvons réécrire **(c)** de la manière suivante :

(exécuter-séquence (LISTOF dernière-action ?ensemble-d-états)) ?état-1) **(c2)**

Par définition de *dernière-action*, le terme *(LISTOF dernière-action ?ensemble-d-états)* correspond à une séquence d'une seule action qui est une action permettant de générer un état appartenant à *?ensemble-d-états*. Comme *?état-1* appartient à l'ensemble des états de départ de *(dernière-action ?ensemble-d-états)*, et que par définition, cet ensemble d'états de départ contient tous les états à partir desquels on peut appliquer *dernière-action* pour produire un état appartenant à *?ensemble-d-états*, on peut en déduire que l'application d'*exécuter-séquence* à *(dernière-action ?ensemble-d-états)* et à *?état-1* (c'est-à-dire le terme **(c2)**) produit un état qui appartient à *?ensemble-d-états*.

---

[49]Pour ne pas alourdir la démonstration principale (déjà fort compliquée), nous ne démontrerons pas cette affirmation. Nous proposons au lecteur de le faire à titre d'exercice. La thèse exacte à démontrer est la suivante :

(=> (= ?l (append ?l1 l2))
(= (exécuter-séquence ?l ?état-de-départ)
(exécuter-séquence ?l2 (exécuter-séquence ?l1 ?état-de-départ))))

Puisque **(c2)** est une autre façon d'écrire le terme **(c)**, qui est lui-même égal à **(b)**, on peut affirmer la proposition ci-dessous :

(member (exécuter-séquence (construire-solution état-initial ?ensemble-d-états)) ?ensemble-d-états)

Ce qui correspond à la définition de la relation *est-solution-générale* et permet d'affirmer la proposition suivante :

(est-solution-générale (construire-solution état-initial-du-problème ?ensemble-d-états)
état-initial-du-problème
?ensemble-d-états)

C.Q.F.D.

6.5. EVALUATION DE LA SOLUTION PROPOSEE

Nous terminons ce chapitre par une évaluation de la solution que nous avons proposée pour représenter l'aspect "métaconnaissance" du problème du singe en KIF. Rappelons une fois encore que si cette évaluation met quelque peu la puissance d'expression du langage KIF en cause, elle n'a toutefois pas un caractère aussi affirmé que celle que nous avons faite à la fin du chapitre 5 pour le reste de la spécification, puisque la solution proposée ici n'est qu'une piste explorée, et que d'autres solutions sont envisageables, en particulier celles qui utiliseraient les concepts de KIF relatifs à la métaconnaissance (voir 3.4.) et tenteraient dès lors une approche plus "classique" de la stratégie "orientée-buts"[50]

Le principal avantage de la solution que nous avons proposée ci-dessus est bien sûr le respect de la stratégie "orientée-buts" (puisque, comme on l'a vu, la séquence d'actions construite respecte le principe général de cette stratégie) tout en adaptant cette stratégie au langage fonctionnel qu'est KIF, alors qu'*a priori* la meilleure description de cette stratégie est de type algorithmique[51] (**tant que** sous-buts non tous résolus **faire** choisir sous-buts à résoudre ; déterminer action ; générer nouveaux sous-buts **fintant**). Or, l'adaptation d'une spécification décrite de façon fonctionnelle à un langage d'implémentation algorithmique est plus immédiat que ne le serait l'adaptation inverse, c'est à dire la traduction d'une spécification algorithmique dans un langage d'implémentation fonctionnel comme LISP.

D'autre part, reconnaissons que l'adaptation de cette stratégie algorithmique à un "moule" fonctionnel et déclaratif ne va pas sans heurts, principalement au niveau du choix de l'action à réaliser : premièrement parce que nous avons dû ramener les critères de choix (heuristiques) à un texte informel, alors que certains de ces critères (par exemple, ne pas re-générer un ensemble d'états déjà préalablement produit) pourraient être écrits formellement dans une description

---

[50]Si nous n'avons pas tenté cette démarche, c'est parce qu'il nous a semblé que le lien entre le concept de but et les notions d'état et d'action ne nous a pas paru représentable. Mais la question de cette représentation reste posée.

[51]Ce qui ne veut pas dire qu'elle est déterministe, puisque le problème du choix reste basé sur des heuristiques

algorithmique de la stratégie ; deuxièmement parce qu'une solution algorithmique permettrait d'envisager le problème du backtracking, c'est-à-dire du retour en arrière lorsque la séquence d'actions déjà générée produit un ensemble d'états impossible à atteindre à partir de l'état de départ. Notre spécification en KIF ignore ce problème, parce qu'elle suppose que le choix de l'action à appliquer sera correct (rappelons que la spécification de ce choix est réduite au strict minimum dans le texte KIF). Notons toutefois que l'aspect "heuristiques de choix" poserait problème également dans une description algorithmique déterministe et qu'il n'a pas été envisagé dans la spécification informelle.

Le deuxième avantage de cette solution est qu'elle est générique pour les problèmes de planification : ceux-ci peuvent tous être décrits en terme d'états et de fonctions de changement d'état, et la spécification de la métaconnaissance que nous avons présentée est pratiquement adaptable à n'importe quel problème de planification, en adaptant toutefois la notion d'*ensemble-des-états-buts* terminaux du problème, puisque ces états dépendent de la notion d'*objet-cible* particulière à ce problème-ci. Une réserve est cependant à ajouter quant à cette généricité : le cas d'un problème dans lequel il pourrait ne pas y avoir de solution (but impossible à atteindre) n'est pas traité ici[52]

Un dernier point à discuter concerne la puissance d'expression de KIF sur le plan de la correction d'une spécification. Au chapitre précédent (point 5.5.2.), nous avions posé la question "Le texte KIF d'une spécification contient-il assez d'information pour permettre d'établir des propriétés de cette spécification ?". Notre réponse avait été "oui", sur base des deux démonstrations du chapitre 5. La démonstration que nous venons de faire à ce chapitre-ci semble infirmer cette réponse : nous avons en effet dû utiliser un texte informel pour valider l'utilisation du principe de récurrence pour cette démonstration. D'un autre côté, rappelons ce que nous avons dit dans l'introduction à ce chapitre : notre proposition de représentation de la métaconnaissance est personnelle, avec les avantages et les inconvénients que nous venons d'établir, et ne nous permet pas de tirer des conclusions définitives sur le langage KIF lui-même en tant que langage de spécification, sinon bien sûr que l'expression de la métaconnaissance n'y est pas aisée. Nuançons donc notre réponse du point 5.5.2. en disant ceci : *dans la mesure où les propriétés à établir dans une spécification KIF portent sur des objets clairement définis dans cette spécification, alors le texte de la spécification contient (ou peut contenir) assez d'information pour permettre l'établissement de ces propriétés.*

En conclusion, nous pensons que la solution de représentation de la métaconnaissance développée dans ce chapitre, bien que bâtarde sur le plan du choix, a des avantages certains mais que KIF manque de concepts pour exprimer de façon plus simple les notions relatives à cette métaconnaissance. Finalement, un bon langage de spécification de systèmes experts pourrait être un langage qui reprendrait les idées de KIF pour la représentation du domaine du problème, et les idées d'un autre langage (peut-être encore inexistant) pour celle de la métaconnaissance.

---

[52]Ce qui ici ne pose pas de problème puisque, comme nous l'avons montré au chapitre 5, il existe toujours au moins une solution au problème du singe.

CONCLUSION

L'adaptation des techniques du génie logiciel au développement des systèmes experts est un domaine encore peu exploré. Ce mémoire s'inscrit dans un tel cadre et prétend donc contribuer - modestement - à l'élaboration d'un "génie des connaissances", qui permettrait de développer les systèmes experts de façon à assurer la complétude et la consistance de ces systèmes, ainsi que leur réutilisabilité.

L'élaboration d'un cycle de vie transformationnel adapté aux systèmes experts et composé d'une analyse des besoins et d'une spécification abstraite de la solution est une étape indispensable pour la mise sur pied de ce "génie des connaissances".

Dans ce cadre de travail, l'étude d'ALBERT, de KIF et des transformations entre les deux langages présentait un intérêt certain. D'une part, ALBERT permet de décrire le problème indépendamment de toute recherche de solution, et semble bien adapté à l'analyse des besoins d'un système expert de planification. D'autre part, sans être un langage de spécification comme Z ou VDM, KIF a l'avantage d'être *formel* et *abstrait* d'une part, et d'être spécifiquement orienté vers la représentation des connaissances d'autre part. Il présentait donc un bon compromis entre le concept de spécification abstraite de la solution (propre au génie logiciel) et celui de représentation des connaissances (spécifique à l'Intelligence Artificielle, et plus précisément aux systèmes experts).

Le travail réalisé n'est pas inutile, puisqu'il a au moins permis de débroussailler le terrain "KIF", et puisque les chercheurs dont le but est de concevoir un langage de spécification pour systèmes experts peuvent puiser dans ce document quelques idées de concepts qu'il peut être intéressant d'inclure à ce langage de spécification. Ceci concerne principalement l'équipe du projet KSL, projet mentionné dans l'introduction de ce document.

Ce travail a aussi la prétention d'avoir apporté une petite amélioration personnelle à KIF en lui donnant une représentation de quelques concepts génériques dans le domaine des problèmes de planification, en particulier la représentation de la solution d'un problème de planification (section 4.7) et une représentation possible de la stratégie d'un tel problème (chapitre 6).

Ce mémoire ne fait toutefois qu'ouvrir la voie à une étude plus poussée du développement des systèmes experts, puisqu'il faudrait encore systématiser les démarches de transformation, et traiter des cas de systèmes experts différents (systèmes d'aide au diagnostic, d'interprétation, de prévision, etc.).

Le cadre de travail est bien large, et encore peu exploré, mais on peut penser que l'avenir des systèmes experts passe par cette adaptation des techniques de développement classiques à leur propre spécificité.

ANNEXE 1 : TEXTE DE LA SPECIFICATION KIF

DEFINITION DES PRODUITS CARTESIENS

```
(DEFOBJECT produit-cartésien-des-objets-physiques :=
    (SETOFALL (LISTOF ?nom ?pos-hor ?pos-ver ?poids)
              (AND (member ?nom (SETOF 'bananes 'lit 'couverture 'échelle))
                   (= ?pos-hor (LISTOF ?abs ?ord))
                   (integer ?abs) (>= ?abs 1) (=< ?abs 10)
                   (integer ?ord) (>= ?ord 1) (=< ?ord 10)
                   (member ?pos-ver (SETOF 'sol 'plafond 'porté @nom2))
                   (= (SETOF @nom2) (SETOF 'bananes 'lit 'couverture 'échelle))
                   (member ?poids 'lourd 'léger))))

(DEFOBJECT produit-cartésien-du-singe :=
    (SETOFALL (LISTOF ?pos-hor ?pos-ver ?objet-porté)
              (AND (= ?pos-hor (LISTOF ?abs ?ord))
                   (integer ?abs) (>= ?abs 1) (=< ?abs 10)
                   (integer ?ord) (>= ?ord 1) (=< ?ord 10)
                   (member ?pos-ver (SETOF 'sol @nom-d-objet))
                   (member ?objet-porté (SETOF 'rien @nom-d-objet))
                   (= (SETOF @nom-d-objet)
                      (SETOF 'bananes 'lit 'couverture 'échelle)))))
```

FONCTIONS D'EXTRACTION D'INFORMATION DANS LES OBJETS DU PROBLEME

```
(DEFFUNCTION nom (?objet) :=
    (IF (member ?objet produit-cartésien-des-objets-physiques)
        (first ?objet)))

(DEFFUNCTION pos-hor (?objet) :=
    (COND ( (member ?objet produit-cartésien-des-objets-physiques) (first (rest ?objet)))
          ( (member ?objet produit-cartésien-du-singe) (first ?objet))))

(DEFFUNCTION pos-ver (?objet) :=
    (COND ((member ?objet produit-cartésien-des-objets-physiques)
           (last (butlast ?objet)))
          ((member ?objet produit-cartésien-du-singe) (first (rest ?objet)))))

(DEFFUNCTION poids (?objet) :=
    (IF (member ?objet produit-cartésien-des-objets-physiques)
        (last ?objet)))

(DEFFUNCTION objet-porté (?singe) :=
    (IF (member ?singe produit-cartésien-des-objets-physiques) (last ?singe)))

(DEFFUNCTION objet-de-nom (?nom-d-objet) :=
    (IF (EXISTS (?objet) (AND (est-un-objet-physique ?objet)
                              (= ?nom-d-objet (nom ?objet))))
        ?objet))
```

INVARIANTS DU PROBLEME

```
(DEFRELATION invariant-ens-obj-1 (?ens-obj) :=
    (FORALL (?nom1)
            (=> (member ?nom (SETOF 'bananes 'lit 'couverture 'échelle))
                (EXISTS (?obj) (AND (member ?obj ?ens-obj)
                                    (= (nom ?obj) ?nom1))))))

(DEFRELATION invariant-ens-obj-2 (?ens-obj) :=
    (FORALL (?obj1 ?obj2)
            (=> (member ?obj1 ?ens-obj)
                (member ?obj2 ?ens-obj)
                (/= ?obj1 ?obj2)
                (/= (nom ?obj1) (nom ?obj2))))))

(DEFRELATION invariant-ens-obj-3 (?ens-obj) :=
    (FORALL (?obj1)
            (=> (member ?obj1 ?ens-obj)
                (OR (= (pos-ver ?obj1) 'plafond)
                    (= (pos-ver ?obj1) 'porté))
                (FORALL (?obj2)
                        (=> (member ?obj2 ?ens-obj)
                            (/= ?obj2 ?obj1)
                            (/= (pos-ver ?obj2) (nom ?obj1)))))))

(DEFRELATION invariant-ens-obj-4 := ...)

...

(DEFRELATION invariant-ens-obj-n := ...)

(DEFRELATION invariant-singe-1 := ...)

...

(DEFRELATION invariant-ens-obj-n := ...)

(DEFRELATION invariant-état-1 := ...)

...

(DEFRELATION invariant-état-n := ...)
```

EXPRESSION DES AGENTS DU PROBLEME ET DE LEURS ETATS

```
(DEFRELATION est-l-ensemble-des-objets-physiques (?ens-obj) :=
    (AND (subset ?ens-obj produit-cartésien-des-objets)
         (invariant-ens-obj-1 ?ens-obj) ... (invariant-ens-obj-n ?ens-obj)))

(DEFRELATION est-un-objet-physique (?obj) :=
    (EXISTS (?ens-obj) (AND (est-l-ensemble-des-objets-physiques ?ens-obj)
                            (member ?obj ?ens-obj))))
```

```
(DEFRELATION est-le-singe (?singe) :=
    (AND (member ?singe produit-cartésien-du-singe)
         (invariant-singe-1 ?singe) ... (invariant-singe-n ?singe)))

(DEFRELATION est-un-état-du-problème (?état) :=
    (EXISTS (?singe ?ens-obj)
            (AND (= ?état LISTOF ?singe ?ens-obj)
                 (est-le-singe ?singe)
                 (est-l-ensemble-des-objets-physiques ?ens-obj)
                 (invariant-état-1 ?état) ... (invariant-état-n ?état))))

(DEFFUNCTION état-du-singe (?état) :=
    (IF (est-un-état-du-problème ?état)
        (first ?état)))

(DEFFUNCTION état-des-objets-physiques (?état) :=
    (IF (est-un-état-du-problème ?état)
        (last ?état)))
```

DEFINITION DE L'ETAT INITIAL DU PROBLEME

```
(DEFOBJECT état-initial-du-problème
    (est-un-état-du-problème état-initial-du-problème))
```

EXPRESSION DES ETATS FINALS DU PROBLEME

```
(DEFOBJECT objet-cible
    (est-un-objet-physique objet-cible)
    (= (poids objet-cible) 'léger))

(DEFOBJECT ensemble-des-états-buts :=
    (SETOFALL (?état-but)
              (AND (est-un-état-du-problème ?état-but)
                   (= (objet-porté (état-du-singe ?état-but)) (nom objet-cible)))))

(DEFRELATION est-état-solution-du-problème (?état) :=
    (member ?état ensemble-des-états-buts))
```

DEFINITION DES FONCTIONS DE CHANGEMENT D'ETAT

```
(DEFFUNCTION saisir-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (poids (objet-de-nom ?nom-obj)) 'léger)
             (= (pos-hor (état-du-singe ?état)) (pos-hor (objet-de-nom ?nom-obj)))
             (FORALL (?obj2) (=> (est-un-objet-physique ?obj2)
                                 (/= (pos-ver ?obj2) ?nom-obj)))
             (OR (AND (= (pos-ver (état-du-singe ?état)) 'sol)
                      (/= (pos-ver (objet-de-nom ?nom-obj)) 'plafond))
                 (AND (= (pos-ver (état-du-singe ?état)) 'échelle)
                      (= (pos-ver (objet-de-nom ?nom-obj)) 'plafond))))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        (pos-ver (état-du-singe ?état))
                        ?nom-obj)
                (union (difference (état-des-objets-physiques ?état)
                                   (SETOF (objet-de-nom ?nom-obj)))
                       (SETOF (LISTOF ?nom-obj
                                      (pos-hor (objet-de-nom ?nom-obj))
                                      porté
                                      (poids (objet-de-nom ?nom-obj)))))))

(DEFFUNCTION lâcher-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (objet-porté (état-du-singe ?état)) ?nom-obj))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        (pos-ver (état-du-singe ?état))
                        'rien)
                (union (difference (état-des-objets-physiques ?état)
                                   (SETOF (objet-de-nom ?nom-obj)))
                       (SETOF (LISTOF ?nom-obj
                                      (pos-hor (objet-de-nom ?nom-obj))
                                      'sol
                                      (poids (objet-de-nom ?nom-obj))))))))
```

```
(DEFFUNCTION aller-en (?état ?x ?y) :=
    (IF (AND (est-un-état-du-problème ?état)
             (integer ?x) (>= ?x 1) (=< ?x 10)
             (integer ?y) (>= ?y 1) (=< ?y 10)
             (= pos-ver (état-du-singe (?état)) 'sol))
        (IF (= (objet-porté (état-du-singe ?état)) 'rien)
            (LISTOF (LISTOF (LISTOF ?x ?y)
                            (pos-ver (état-du-singe ?état))
                            (objet-porté (état-du-singe ?état)))
                    (état-des-objets-physiques ?état))
            (LISTOF (LISTOF (LISTOF ?x ?y)
                            (pos-ver (état-du-singe ?état))
                            (objet-porté (état-du-singe ?état)))
                    (union (difference (état-des-objets-physiques ?état)
                                       (SETOF
                                        (objet-de-nom (objet-porté (état-du-singe ?état)))))
                           (SETOF (LISTOF (objet-porté (état-du-singe ?état))
                                          (LISTOF ?x ?y)
                                          'porté
                                          (poids (objet-de-nom
                                                  (objet-porté
                                                   (état-du-singe ?état))))))))))

(DEFFUNCTION mettre-objet-en (?état ?nom-obj ?x ?y) :=
    (lâcher-objet (aller-en ?état ?x ?y) ?nom-obj))

(DEFFUNCTION monter-sur-objet (?état ?nom-obj) :=
    (IF (AND (est-un-état-du-problème ?état)
             (est-un-objet-physique (objet-de-nom ?nom-obj))
             (= (pos-ver (état-du-singe ?état)) 'sol)
             (= (pos-ver (objet-de-nom ?nom-obj)) 'sol)
             (= (pos-hor (objet-de-nom ?nom-obj)) (pos-hor (état-du-singe ?état)))
             (= (objet-porté (état-du-singe ?état)) 'rien))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        ?nom-obj
                        (objet-porté (état-du-singe ?état)))
                (état-des-objets-physiques ?état))))

(DEFFUNCTION sauter-sur-sol (?état) :=
    (IF (AND (est-un-état-du-problème ?état)
             (/= (pos-ver (état-du-singe ?état)) 'sol))
        (LISTOF (LISTOF (pos-hor (état-du-singe ?état))
                        'sol
                        (objet-porté (état-du-singe ?état)))
                (état-des-objets-physiques ?état))))
```

EXPRESSION DE LA SOLUTION

```
(DEFRELATION est-une-séquence-d-actions (?seq) :=
    (AND (list ?seq)
         (FORALL (?action)
                 (=> (item ?action ?seq)
                     (AND (double ?action)
                          (est-une-fonction-du-problème (first ?action))
                          (EXISTS (?élément-de-fonction)
                                  (AND (member ?élément-de-fonction (first ?action))
                                       (= (rest (butlast ?élément-de-fonction))
                                          (last ?action)))))))))

(DEFRELATION est-une-fonction-du-problème (?fonction) :=
    (member ?fonction (SETOF saisir-objet lâcher-objet aller-en
                             mettre-objet-en monter-sur-objet sauter-sur-sol)))

(DEFFUNCTION exécuter-séquence (?seq ?état-de-départ) :=
    (IF (AND (est-une-séquence-d-actions ?seq)
             (est-un-état-du-problème ?état-de-départ))
        (IF (null ?seq)
            ?état-de-départ
            (exécuter-séquence (rest ?seq)
                               (apply (first (first ?seq))
                                      (append (LISTOF ?état-de-départ)
                                              (last (first ?seq))))))))

(DEFRELATION est-solution-du-problème (?seq) :=
    (est-état-solution-du-problème (exécuter-séquence ?seq état-initial-du-problème)))
```

EXPRESSION DE LA METACONNAISSANCE

```
(DEFFUNCTION fonctions-génératrices (?ensemble-d-états) :=
    (SETOFALL (?f)
              (AND (est-une-fonction-du-problème ?f)
                   (EXISTS (?état @args)
                           (AND (NOT (member ?état ?ensemble-d-états))
                                (member (apply ?f (LISTOF ?état @args))
                                        (?ensemble-d-états)))))))

(DEFFUNCTION meilleure-fonction (?ensemble-de-fonctions ?ensemble-d-états)
    (member (meilleure-fonction (?ensemble-de-fonctions ?ensemble-d-états))
            ?ensemble-de-fonctions))

* HEURISTIQUE *
* Si le singe tient un objet, lui faire lâcher le plus vite possible, donc choisir la *
* fonction (lâcher-objet) le plus tard possible (puisque la solution se construit en *
* commençant par la fin).*

*HEURISTIQUE IMPORTANTE *
* Il faut éviter de choisir une fonction qui produise un ensemble d'états de départ déjà généré
précédemment, ou un sous-ensemble de celui-ci, pour éviter le bouclage. *
```

```
(DEFFUNCTION arguments-générateurs (?f ?ensemble-d-états) :=
    (SETOFALL (LISTOF @args)
              (EXISTS (?état)
                      (member (apply ?f (LISTOF ?état (LISTOF ?@args))
                                     ?ensemble-d-états)))))

(DEFFUNCTION meilleure-liste-d-args (?f ?ensemble-de-listes-d'args ?ensemble-d-états)
    (member meilleure-liste-d-args ?ensemble-de-listes-d'args)

* HEURISTIQUES *
* <au choix de l'expert> *

(DEFFUNCTION dernière-action (?ensemble-d-états) :=
    (IF (AND (EXISTS (?f)
                     (= ?f (meilleure-fonction (fonctions-génératrices ?ensemble-d-états)
                                               ?ensemble-d-états)))
             (EXISTS (LISTOF @args)
                     (= LISTOF @args
                        (meilleure-liste-d-args ?f
                                                (arguments-générateurs ?f
                                                                       ?ensemble-d-états)
                                                ?ensemble-d-états))))
        (LISTOF ?f (LISTOF @args))))

(DEFFUNCTION ensemble-des-états-de-départ (?action ?ensemble-des-états-finals) :=
    (IF (EXISTS (?f @args)
                (AND (est-une-fonction-du-problème ?f)
                     (= (LISTOF ?f (LISTOF @args)) ?action)))
        (SETOFALL (?état)
                  (member (apply ?f (LISTOF ?état @args)) ?ensemble-des-états-finals))))

(DEFFUNCTION construire-solution (?état-initial ?ensemble-d-états-buts) :=
    (IF (member ?état-initial ?ensemble-d-états-buts)
        (LISTOF)
        (append (construire-solution ?état-initial
                                     (ensemble des états de départ
                                                (dernière-action ?ensemble-d-états-buts)
                                                ?ensemble-d-états-buts))
                (LISTOF (dernière-action ?ensemble-d-états-buts)))))

(DEFOBJECT solution-optimale-du-problème :=
    (construire-solution état-initial-du-problème ensemble-des-états-buts))
```

PROPOSITIONS DE CONSISTANCE

```
* Consistance des fonctions de changement d'état *

(FORALL ( ?f ?état @arguments)
    (=> (est-une-fonction-du-problème ?f)
        (/= ⊥ (apply ?f (LISTOF ?état @arguments)))
        (est-un-état-du-problème (apply ?f (LISTOF?état @arguments)))))
```

```
* Existence d'une solution *

(FORALL (?état-initial ?objet-cible)
        (=> (est-un-état-du-problème ?état-initial)
            (est-un-objet-physique ?objet-cible)
            (= (poids ?objet-cible) 'léger)
            (EXISTS (?seq)
                    (member
                        (exécuter-séquence ?seq ?état-initial)
                        (SETOFALL (?état)
                                  (AND (est-un-état-du-problème ?état)
                                       (= (objet-porté (état-du-singe?état))
                                          (nom ?objet-cible))))))))

* Correction de la solution optimale (obtenue par la stratégie orientée-buts) *

(est-solution-du-problème solution-optimale-du-problème)

...
```

CONFIGURATION INITIALE DU PROBLEME (PARTIE DYNAMIQUE)

```
(DEFOBJECT état-initial-du-problème
    (= (état-du-singe état-initial-du-problème)
       (LISTOF (LISTOF 5 10) 'sol 'couverture))
    (member (LISTOF 'bananes (LISTOF 2 3) 'plafond 'léger)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'lit (LISTOF 1 1) 'sol 'lourd)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'couverture (LISTOF 5 10) 'porté 'léger)
            (état-des-objets-physiques état-initial-du-problème))
    (member (LISTOF 'échelle (LISTOF 1 1) 'lit 'léger)
            (état-des-objets-physiques état-initial-du-problème)))

(DEFOBJECT objet-cible
    (= (nom objet-cible) 'bananes))
```

ANNEXE 2 : SYNTAXE BNF DE KIF

MOTS ET EXPRESSIONS

<word> ::= objet syntaxique de base
<expression> ::= <word> | (<expression>*)

CATEGORIES DE MOTS

Variables

<variable> ::= <indvar> | <seqvar>
<indvar> ::= *un mot commencant par ?*
<seqvar> ::= *un mot commencant par @*

Opérateurs

<operator> ::= <termop> | <sentop> | <ruleop> | <defop>
<termop> ::= LISTOF | SETOF | QUOTE | IF | COND | THE | SETOFALL | KAPPA | LAMBDA
<sentop> ::= = | /= | NOT | AND | OR | => | <= | <=> | forall | exists
<ruleop> ::= =>> | <<= | CONSIS
<defop> ::= DEFOBJECT | DEFFUNCTION | DEFRELATION | := | :=> | :&

Constantes

<constant> ::= <objconst> | <funconst> | <relconst> | <logconst>
<objconst> ::= *un mot désignant un objet*
<funconst> ::= *un mot désignant une fonction*
<relconst> ::= *un mot désignant une relation*
<logconst> ::= *un mot désignant une valeur de vérité*

TERMES

Syntaxe générale

<term> ::= <indvar> | <objconst> | <funconst> | <relconst> | <funterm> | <listterm> | <setterm> | <quoterm> | <logterm> | <quantterm>

Terme de fonction

<funterm> ::= (<funconst> <term>* [<seqvar>])

Terme de liste

<listterm> ::= (LISTOF <term>* [<seqvar>])

Terme d'ensemble

<setterm> ::= (SETOF <term>* [<seqvar>])

Terme quoté

<quoterm> ::= (QUOTE <expression>)

Terme logique

<logterm> ::= (IF <sentence> <term> [<term>]) | (COND (<sentence><term>)*)

Terme quantifié

<quantterm> ::= (THE <term> <sentence>) | (SETOFALL <term> <sentence>) |
(KAPPA (<indvar>* [<seqvar>]) <sentence>) |
(LAMBDA (<indvar>* [<seqvar>]) <term>)

PROPOSITIONS

Syntaxe générale

<sentence> ::= <logconst> | <equation> | <inequality> | <relsent> |
<logsent> | <quantsent>

Egalité

<equation> ::= (= <term> <term>)

Inégalité

<inequality> ::= (/= <term> <term>)

Proposition relationnelle

<relsent> ::= (<relconst> <term>* [<seqvar>]) |
(<funconst> <term>* <term>)

Proposition composée

<logsent> ::= (NOT <sentence>) | (AND<sentence> <sentence>*) |
(OR<sentence><sentence>*) |(=> <sentence>*<sentence> <sentence>) |
(<= <sentence><sentence> <sentence>*) |(<=> <sentence> <sentence>)

Proposition quantifiée

<quantsent> ::= (FORALL <indvar> <sentence>) |
(FORALL (<indvar>*[<seqvar>]) <sentence>) | (EXISTS <indvar> <sentence>)|
(EXISTS (<indvar>*[<seqvar>]) <sentence>)

DEFINITIONS

Syntaxe générale

<definition> ::= <complete> | <partial>
<partial> ::= <conservative> | <unrestricted>

Définition complète

<complete> ::= (DEFOBJECT <objconst> := <term>) |
(DEFFUNCTION <funconst> (<indvar>* [<seqvar>]) := <term>) |
(DEFRELATION <relconst> (<indvar>* [<seqvar>]) := <sentence>)

Définition partielle conservatrice

<conservative> ::=
(DEFOBJECT <objconst> [:conservative-axiom <sentence>]) |
(DEFFUNCTION <funconst> [:conservative-axiom <sentence>]) |
(DEFRELATION <relconst> [:conservative-axiom <sentence>]) |
(DEFRELATION <relconst> (<indvar>* [<seqvar>]) :=>
<sentence> [:conservative-axiom <sentence>])

Définition partielle non restreinte

<unrestricted> ::=
(DEFOBJECT <objconst> <sentence>*) |
(DEFFUNCTION <funconst> <sentence>*) |
(DEFRELATION <relconst> <sentence>*) |
(DEFRELATION <relconst> (<indvar>* [<seqvar>]) :=>
<sentence> [:axiom <sentence>])

REGLES

<rule> ::= (=>> <premise>* <sentence>) | (<<= <sentence> <premise>*)
<premise> ::= <sentence> | (CONSIS <sentence>)

TEXTE KIF

<form> ::= <sentence> | <definition> | <rule>
<knowledge-base> ::= <form>*

BIBLIOGRAPHIE

- **[BLONDIAU93]** J.-F. Blondiau, "Spécification de systèmes experts : évaluation de KIF", rapport de stage effectué dans le cadre du projet KSL (contrat n°3437.00/DERI), Centre d'Etudes et de Recherches de Toulouse, janvier 1993
- **[BUSSENOT92]** J.-L. Bussenot, J. Foisseau & M. Lemoine, "Un exemple de modélisation informelle", rapport technique rédigé dans le cadre du projet KSL (contrat n° 3437.00/DERI), Centre d'Etudes et de Recherches de Toulouse, août 1992
- **[DUBOIS91]** E.Dubois, Ph. Du Bois, A. Rifaut & P.Wodon, "GLIDER User Manual", ESPRIT Projet Icarus 2537, juin1991
- **[DUBOIS93]** E.Dubois, Ph. Du Bois & M. Petit, "O-O Requirement Analysis : an Agent Perspective", à paraître dans "European Conference on Object Oriented Programming", juillet 1993
- **[GENESERETH92]** M. Genesereth & R. Fikes, "Knowledge Interchange Format Version 3.0 Reference Manual", Logic Group, Report Logic 92-1, Computer Science Department, Stanford University, Stanford (U.S.A.), juin 1992
- **[GEORGEFF87]** M. Georgeff, "Planning", article paru dans "Annual Reviews of Computer Science 1987", 1987
- **[JACKSON90]** P.Jackson, "Introduction to expert systems", deuxième édition, Adisson Wesley Publishing, 1990